UNIVERZITA PARDUBICE

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Simulace a transformace v umělecké tvorbě Charles Chuck Csuri

Bakalářská práce

2025 Patrik Novotný

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Patrik Novotný**
Osobní číslo:	**I21202**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Simulace a transformace v umělecké tvorbě Charles Chuck Csuri**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Vývoj výpočetní techniky a počítačové grafiky v 60. letech 20. století poskytl umělcům a výzkumníkům prostor pro nový umělecký směr: digitální umění. Simulovaná náhodná čísla, účastníkem zadané vstupní údaje a nastavená pravidla umožnily na počítači zkonstruovat grafické výstupy, zvuky a kinetické události. K pionýrům digitálního umění patřil Charles Chuck Csuri (1922-2022). Jeho grafická tvorba byla často založena na generování náhodných čísel a transformacích. První jeho algoritmy implementoval programátor James Shaffer ve Fortranu na počítači IBM 7094.
Cílem práce dohledat či odhadnout přístupy, které Csuri využil ve své tvorbě, např. při konstrukci obrazů Flies, Flies Transformed, From square to circle (vitruvian), Random War of soldiers. Tyto metody budou tvořit základ aplikace, která obdobné obrazy dokáže nasimulovat a vykreslit.
Součástí naprogramované aplikace bude uživatelská příručka.

Rozsah pracovní zprávy: **45**
Rozsah grafických prací: **5**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

MAREK, J., NEDVĚDOVÁ, M. Historie digitálního umění: Náhoda, počítač a Linie Zdeňka Sýkory. In 37. mezinárodní konference Historie matematiky. Praha: Matfyzpress, 2016. s. 137-146. ISBN 978-80--7378-317-4.
CSURI, C., SHAFFER J. Art, computers and mathematics. In: Murphy J. E. (ed.): Fall Joint Computer Conference AFIPS '68, Thomson Book Company, Washington, 1968, str. 1293–1298.

Vedoucí bakalářské práce: **Mgr. Jaroslav Marek, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlašuji:

Práci s názvem Simulace a transformace v umělecké tvorbě Charles Chuck Csuri jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 14. 5. 2025

Patrik Novotný

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce panu Mgr. Jaroslavu Markovi Ph.D. za odborné vedení, trpělivé konzultace a cenné připomínky, které významně přispěly k úspěšnému dokončení této práce.

Děkuji svým přátelům, spolužákům i rodině za neustálou podporu, inspirativní diskuse a trpělivé povzbuzení, které mi pomohly rozšířit obzory v oblasti generativního umění a dodávaly mi sílu úspěšně dokončit tuto práci.

ANOTACE

Digitální umění spojuje vizuální tvorbu s výpočetními metodami. Tato práce zkoumá tvorbu Charlese Chucka Csurího, průkopníka generativního umění, a zaměřuje se na její algoritmizaci. V úvodní části je popsán historický vývoj počítačové grafiky a technologií, které umožnily vznik digitálního umění. Pozornost je věnována nejznámějším autorům digitálního uměni. Teoretická část práce prezentuje matematickou reprezentaci Csuriho algoritmických postupů, kde klíčovou roli mají zejména pseudonáhodné generování čísel a afinní transformace. V praktické části je vytvořen návrh a prezentována realizace simulační aplikace, která replikuje díla *Random War*, *Flies* a *Flies Transformed*. Uživatel má možnost upravovat vstupní parametry pro generování obrazů a obrazy modifikovat. podle vlastních představ.

KLÍČOVÁ SLOVA

digitální umění, Charles Chuck Csuri, obraz Random War, obraz Flies, obraz Flies Transformed, pseudonáhodné generování čísel

TITLE

Simulation and Transformation in the Artwork of Charles Chuck Csuri

ANNOTATION

Digital art combines visual creation with computational methods. This thesis explores the work of Charles Chuck Csuri, a pioneer of generative art, and focuses on its algorithmization. The introductory section describes the historical development of computer graphics and the technologies that enabled the emergence of digital art. Attention is paid to the most famous digital artists. The theoretical part of the thesis presents a mathematical representation of Csuri's algorithmic procedures, where pseudo-random number generation and affine transformations play a key role. In the practical part, the design and implementation of a simulation application that replicates the works of Random War, Flies and Flies Transformed is developed and presented. The user has the possibility to modify the input parameters for generating images and to modify the images. The user is able to modify and modify the images according to his/her own ideas.

KEYWORDS

digital art, Charles Chuck Csuri, painting Random War, painting Flies, painting Flies Transformed, pseudo-random number generation

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

ÚVOD

Digitální umění přineslo v druhé polovině 20. století revoluci v oblasti umělecké tvorby. Tento nový umělecký směr byl umožněn především díky pokrokům ve výpočetní technice, která nasměřovala umělce k experimentování s novými formami vizuálního vyjádření Zatímco tradiční umělecké techniky, jako je malba, sochařství nebo grafika, byly dlouhodobě definovány fyzickými médii a manuálními postupy, digitální umění nabízí novou dimenzi tvorby, kde se setkává technologie s kreativitou. Umělci získali nástroje, které jim umožňují nejen vytvářet nové druhy vizuálních děl, ale také zkoumat a manipulovat s koncepty, jako jsou náhodnost, transformace a simulace. Tyto pojmy, které by byly v tradičním umění obtížně realizovatelné, se staly klíčovými prvky digitálního umění.

Významným představitelem digitálního umění je Charles Chuck Csuri, jehož tvorba a inovativní přístupy významně přispěly k formování tohoto uměleckého směru. Csuri je často označován za „otce počítačového umění“. Jeho práce spojují uměleckou kreativitu s výpočetní technikou, což mu umožnilo vytvářet díla s vizuální komplexností.

Motivace pro výběr tohoto tématu tedy pramení z fascinace způsobem, jakým Csuri dokázal využít technologie k vytvoření nových uměleckých forem a z touhy prozkoumat jeho metody a techniky v kontextu současného digitálního umění.

Digitální umění se stalo významnou a inovativní kapitolou v dějinách umění, jehož vývoj je úzce spjat s rozvojem výpočetní techniky. Tradiční umělecké disciplíny, jako jsou malba, sochařství nebo grafika, mají kořeny sahající hluboko do minulosti. Zatímco digitální umění je relativně novým fenoménem, který se začal formovat až s příchodem prvních počítačů. Tyto stroje otevřely umělcům zcela nové možnosti vyjádření, které byly dříve nemyslitelné. Abychom porozuměli vzniku a rozvoji digitálního umění, je nezbytné se zaměřit na historický kontext, v němž se tento druh umění vyvíjel.

Vznik digitálního umění je neodmyslitelně spjat s vývojem výpočetní techniky. První počítače byly původně konstruovány především pro vědecké a vojenské účely. Jejich schopnosti byly v počátcích velmi omezené. Avšak jak se technologie rozvíjela, někteří vizionáři začali objevovat potenciál počítačů pro uměleckou tvorbu. Zásadní zlom nastal v 60. letech 20. století, kdy se objevily první počítače schopné zpracovávat grafická data. To položilo základy digitálního umění, které se rychle rozvíjelo díky rostoucím možnostem výpočetní techniky.

Jedním z klíčových technologických pokroků v této době bylo uvedení počítače IBM 7094, který byl představen v roce 1962. Tento stroj, vyvinutý společností IBM, byl jedním z nejvýkonnějších počítačů své doby a jeho schopnost provádět složité matematické operace a zpracovávat velké množství dat umožnila vytváření prvních počítačově generovaných obrazů. Tento technologický milník byl pro umělce klíčový, neboť jim umožnil experimentovat s novými formami vizuálního vyjádření. Počítače jako IBM 7094, ale i další, jako například PDP-1 od společnosti Digital Equipment Corporation, se staly nástroji, které umělci využívali k průkopnickým experimentům v oblasti počítačové grafiky.

Šedesátá léta byla obdobím rychlého rozvoje výpočetní techniky a počítačové grafiky. Pokrok počítačové grafiky byl jedním z klíčových faktorů, který umožnil vznik digitálního umění.

V této době začaly vznikat první specializované programy a algoritmy, které umožňovaly vytváření grafických obrazů na základě matematických modelů a výpočetních procesů. Počítačová grafika se rychle vyvíjela a umělci, kteří se zajímali o nové technologie, začali objevovat její potenciál pro uměleckou tvorbu. Výpočetní technika v 60. letech tedy nejenže otevřela nové možnosti pro vědecký výzkum, ale také položila základy pro vznik nového uměleckého směru – digitálního umění.

První pokusy o využití počítačů k umělecké tvorbě se objevily na přelomu 50. a 60. let 20. století. Jedním z prvních průkopníků byl Ben Laposky, který ve 40. letech začal vytvářet oscilografické obrazy. Laposky prováděl experimenty s elektronickými obvody generující světelné vlny na osciloskopu, které byly fotografovány a výsledkem byly abstraktní vizuální kompozice. I když jeho práce nebyla přímo digitální, ukázala možnosti využití elektronických zařízení k tvorbě umění.

V 60. letech se však objevili první skuteční digitální umělci, kteří začali využívat počítače k tvorbě uměleckých děl. Mezi nimi byl například Frieder Nake, německý matematik a umělec, který vytvořil první počítačově generované obrazy pomocí algoritmů a pseudonáhodných generátorů čísel. Jeho díla se stala jedním z prvních příkladů digitálního umění v Evropě. Dalším pionýrem digitálního umění byl americký umělec Charles Chuck Csuri, který je často označován za „otce počítačového umění“. Csuri začal experimentovat s počítačovou grafikou v polovině 60. let a jeho práce rychle získala uznání pro svůj inovativní přístup k využití technologie v umění. Csuriho díla byla charakteristická využitím pseudonáhodných čísel a algoritmů, které mu umožnily vytvářet složité vizuální struktury, jež by byly tradičními metodami prakticky nedosažitelné.

První digitální umělci, jako byli Frieder Nake, Ben Laposky a Charles Chuck Csuri, sehráli klíčovou roli v definování nového uměleckého směru, který kombinoval matematiku, algoritmy a estetiku. Tito umělci byli mezi prvními, kteří viděli potenciál počítačů jako nástroje pro uměleckou tvorbu. Jejich práce otevřely dveře pro další generace umělců, kteří následně využili digitální technologie k vytváření stále sofistikovanějších děl. V době, kdy byly počítače považovány především za nástroje pro vědecké a průmyslové účely, tito průkopníci ukázali, že mohou sloužit také jako prostředky pro kreativní vyjádření.

Digitální umění se tak stalo důležitým příspěvkem k modernímu umění a ukázalo, že technologie může být mocným nástrojem pro uměleckou tvorbu.

Nejznámější autoři digitálního umění v České republice

Zdeněk Sýkora

Český malíř a grafik Zdeněk Sýkora (1920–2011) patří k prvním evropským umělcům, kteří využili počítač pro uměleckou tvorbu. V letech 1962–1963 experimentoval s *Šedou strukturou* a ve spolupráci s Jaroslavem Blažkem na počítači LGP-30 vytvořil první programované makrostruktury (1964). Sýkorův přístup ke strukturám a makrostrukturám představuje klíčový příspěvek k propojení geometrie. Jeho lineární programy umožnily systematickou variaci tvaru a rozmístění prvků v pravidelné čtvercové síti, když o umístění elementů rozhodovalo počáteční umístění části elementů a o zbývajících rozhodla vybraná

pravidla a náhodnost. Algoritmus generování struktur Zdeněk Sýkora publikoval společně s matematikem Blažkem v americkém časopise Leonardo.[1] Podrobněji také viz [2], [3].

Od roku 1971 vytvářel liniové obrazy, které rozvíjel až do konce života. Sýkora algoritmus pro konstrukci linií založený na využití pseudonáhodných čísel částečně a neúplně publikoval v knize Zdeněk Sýkora 90 [4]. Podrobnější popis jeho geometrické konstrukce linií je publikován v [5, 6, 7].

Bohuslav "Woody" Vašulka

Woody Vašulka (1937–2019), český průkopník videoartu, představil v roce 1967 videoinstalace pro Americký pavilon na Expo 67 v Montrealu. V následujících letech pracoval s vlastním přístrojem Digital Image Articulator, jehož ukázky jako *Vocabulary* (1973) či *The Matter* (1974) patří k raným evropským dílům hybridního digitálně-video artového charakteru.[8]

Nejznámější autoři digitálního umění ve světě

Ben F. Laposky

Jako vůbec první experimentátor s elektronickými zařízeními vytvářel Laposky v roce 1950 na osciloskopu sérii abstraktních obrazů nazvanou *Oscillons* neboli „Electronic Abstractions". Tyto kompozice, vzniklé pomocí analogových oscilátorů, byly fotograficky zachyceny. Jeho díla byla v roce 1952–1953 vystavena v Sanford Museum. Ve své tvorbě formuloval ideje pro vizuální uměleckou tvorbu řízenou elektronikou.[9]

A. Michael Noll

V létě 1962 koncipoval americký vědec Michael Noll dílo *Gaussian–Quadratic*, na kterém během roku 1963 pracoval na mainframu IBM. Výsledná grafika založená na matematických funkcích byla v roce 1965 poprvé veřejně prezentována a chráněna symbolem autorských práv (copyrightem), což ji řadí mezi první oficiálně uznaná počítačově generovaná umělecká díla.[10]

Frieder Nake

Německý matematik a umělec Frieder Nake začal během roku 1963 tvořit první generativní kresby pomocí pseudonáhodných algoritmů. Jeho práce získaly širší pozornost na výstavě „Computer-Grafik" v galerii Wendelin Niedlich ve Stuttgartu v listopadu 1965, kde byly poprvé představeny evropské experimenty s počítačovou grafikou.[11]

Georg Nees

Další německý průkopník Georg Nees experimentoval s počítačovou grafikou od poloviny 60. let. Jeho dílo *K27* (1966) a následující *Schotter* neboli *Cubic Disarray* (1968) ukázala, jak lze pomocí jednoduchých matematických pravidel vytvořit složité vizuální struktury, jež balancují na hranici řádu a chaosu.[12]

Vera Molnár

Vera Molnár, narozená v Maďarsku a později působící ve Francii, vytvořila v letech 1968–1969 sérii plotterových kreseb *Interruptions*, které byly založeny na drobných variacích v pravidelných geometrických vzorech, čímž zdůraznila kontrast mezi opakováním a náhodou v algoritmickém umění.[13]

Manfred Mohr

Německý umělec žijící a tvořící v USA Manfred Mohr začal koncem 60. let experimentovat s plotrem a vytvořil řadu abstraktních černobílých kompozic. Jeho první sólová výstava „Une Esthétique Programmée“ proběhla v roce 1971 v Musée d'Art Moderne de la Ville de Paris.[14]

Harold Cohen

Britský umělec Harold Cohen vyvinul v roce 1973 autonomní program AARON, který samostatně generoval kresby na základě zabudovaných pravidel a heuristik. Program se postupně vyvíjel až do počátku 2010 let a patří k nejdéle fungujícím počítačovým uměleckým systémům.[15]

Lillian Schwartz

Americká vizuální umělkyně Lillian Schwartz je známá především svými ranými experimenty s počítačovou animací, například *Pixillation* (1971) a *Olympiad* (1972). Její práce kombinovaly film, video a digitální techniku, čímž rozšířily hranice multimediálního umění.[16]

Charles Csuri

Charles Chuck Csuri, často označovaný jako „otec počítačového umění“, je postava, která sehrála klíčovou roli v rozvoji digitálního umění a výpočetní grafiky. Jeho život a kariéra jsou úzce spjaty s vývojem této oblasti, která přinesla revoluční změny do světa umění. Csuriho příběh je příběhem umělce, který se z tradičního malíře stal průkopníkem digitálního umění. Jeho cesta je plná inovací, objevů a překonávání hranic mezi uměním a technologií.

Charles Chuck Csuri se narodil 4. července 1922 v Zanesville, Ohio. Od mládí projevoval zájem o umění, což ho přivedlo ke studiu na Ohio State University, kde získal titul bakaláře a poté magistra umění. Během svého studia se věnoval především tradičnímu malířství a kresbě. Jeho raná tvorba byla silně ovlivněna modernistickými směry, jako je abstraktní expresionismus. Nicméně jeho kariéra byla přerušena druhou světovou válkou, kdy od roku 1943 do roku 1946 sloužil v armádě Spojených států a zúčastnil se bojů v Evropě, v roce 1945 obdržel Bronzovou hvězdu za hrdinství v bitvě v Ardenách. Po válce se vrátil do Spojených států a pokračoval ve své umělecké dráze.

Po skončení války se Csuri vrátil na Ohio State University, kde se začal zajímat o možnosti, které nabízela nově se rozvíjející výpočetní technika. V roce 1964 se rozhodl prozkoumat využití počítačů pro uměleckou tvorbu, což bylo v té době radikální rozhodnutí. V době, kdy počítače byly stále relativně novým fenoménem a byly využívány především pro vědecké a technické účely, Csuri rozpoznal jejich potenciál jako nástroje pro tvorbu umění. Tento zájem

ho přivedl ke spolupráci s výzkumníky z oblasti informatiky a matematiky, což vedlo k vytvoření jednoho z prvních interdisciplinárních týmů zaměřených na výzkum počítačového umění.

Csuriho umělecké techniky byly založeny na jeho schopnosti využívat počítač jako nástroj nejen pro tvorbu, ale také pro experimentování s koncepty, které by byly v tradičním umění obtížně realizovatelné. Jedním z hlavních nástrojů, které Csuri ve své tvorbě používal, byly pseudonáhodné generátory čísel (PRNG). Tyto generátory mu umožnily zavádět náhodnost do procesu tvorby, což vedlo k vytváření vizuálních kompozic, které byly jak nečekané, tak i vysoce komplexní. Náhodnost se pro Csuriho stala způsobem, jak obohatit umění o nové dimenze a jak dosáhnout efektů, které by byly tradičními metodami prakticky nedosažitelné.

Dalším klíčovým prvkem Csuriho uměleckého arzenálu byly matematické transformace. Tyto operace, které měnily základní parametry grafických prvků – jako je jejich velikost, tvar nebo orientace – umožnily Csurimu vytvářet složité vizuální struktury, které se často nacházely na pomezí abstrakce a reality. Transformace mu dovolily experimentovat s možnostmi, jak může být umělecké dílo v čase dynamicky měněno a jak mohou být jednoduché formy přetvořeny do něčeho nečekaného a vizuálně fascinujícího.

Tento inovativní přístup se jasně projevuje v některých z jeho nejznámějších děl. Například dílo „Flies“ z roku 1967 je ukázkou toho, jak Csuri využíval náhodnost k dosažení složitých kompozic. Grafické prvky reprezentující mouchy byly na plátně rozmístěny podle náhodně generovaných hodnot, což vedlo k vytvoření scény plné dynamiky a pohybu.

V dalším díle, „Flies Transformed“ (1968), Csuri posunul své experimenty ještě dál. Pomocí transformací měnil původní grafické prvky tak, že výsledný obraz byl zcela odlišný od výchozího stavu. Tato práce ukazuje, jak může být umělecké dílo neustále přetvářeno a jak proces tvorby může být stejně důležitý jako samotný konečný výsledek. Csuri tímto dílem demonstroval, že umění může být živým a neustále se vyvíjejícím procesem, což bylo v té době revoluční myšlenkou.

„Random War of Soldiers“ (1967) je další významné dílo, které ilustruje Csuriho mistrovské zvládnutí náhodnosti a generativního designu. V tomto díle vytvořil obraz, kde postavy vojáků byly rozmístěny na bojišti podle náhodně generovaných hodnot. Výsledkem je obraz, který na první pohled působí jako zachycení chaotického boje, přestože byl pečlivě zkonstruován pomocí matematických algoritmů. Toto dílo je ukázkou toho, jak může být náhodnost využita k vytvoření scény, která nejenže odráží složitost reálného světa, ale také zpochybňuje tradiční představy o tom, co může být považováno za umění.

Csuriho přístup k digitálnímu umění byl unikátní tím, že v něm spojoval tradiční umělecké techniky s novými technologiemi. Na jedné straně byl stále malířem, který vnímal estetiku, kompozici a barvu jako klíčové prvky svého díla. Na druhé straně byl inovátorem, který využíval počítače a algoritmy k dosažení výsledků, které by byly tradičními metodami nemožné. Tato kombinace z něj udělala průkopníka, který výrazně ovlivnil další generace umělců a vědců.

Jeho kariéra však nebyla omezena pouze na vlastní tvorbu. Csuri také sehrál klíčovou roli jako pedagog a mentor, který své znalosti a zkušenosti předával dalším generacím. Během své dlouhé kariéry na Ohio State University založil a vedl několik výzkumných center, včetně Computer Graphics Research Group, které se staly líhní pro mnoho významných osobností v oblasti digitálního umění a počítačové grafiky. Jeho pedagogický přístup byl charakteristický tím, že studenty povzbuzoval k experimentování a objevování nových cest, jak propojit umění s technologií.

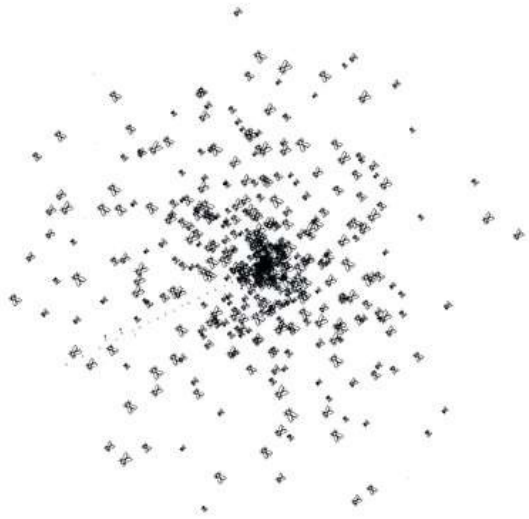
Charles Chuck Csuri zemřel 27. února 2022 ve věku 99 let, ale jeho odkaz žije dál. Jeho díla jsou součástí významných světových sbírek a jsou vystavována v prestižních galeriích a muzeích po celém světě. Csuriho vliv na digitální umění je nezpochybnitelný, a jeho přístup k umělecké tvorbě inspiroval a nadále inspiruje umělce, kteří se zabývají propojením umění a technologie. Jeho život a dílo jsou příkladem toho, jak může umělec překročit hranice svého oboru a otevřít nové cesty pro kreativní vyjádření.

Významná Csuriho díla:

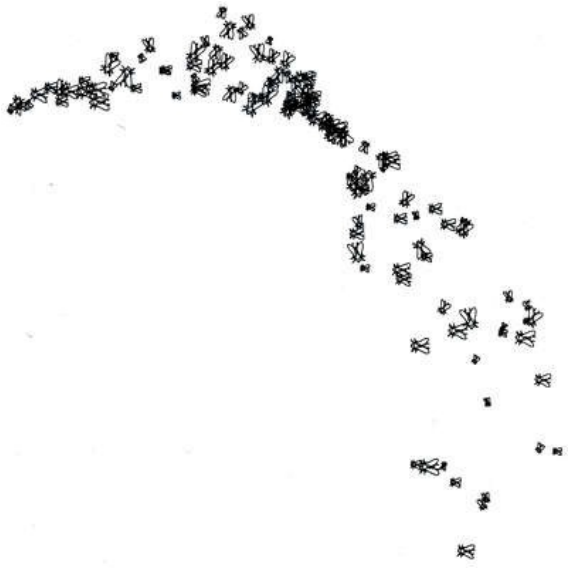
Flies, 1966

V počáteční fázi své umělecké kariéry se Charles Csuri věnoval experimentům s náhodností, kterou využíval jako klíčový princip při generování vizuálních kompozic. Pomocí generátoru náhodných čísel určoval například pozice much v práci *Feeding Time* (1966) nebo rozmístění figurek vojáků v díle *Random War* (1967).[17]

V období let 1966–1967 se motiv mouchy domácí opakovaně objevoval v Csurim díle, často v rámci děl s ironickým nebo lehce konspirativním podtextem, které reflektovaly autorův smysl pro hravost a poetiku absurdity. V případě *Feeding Time* byla poloha, velikost a orientace každé mouchy určována stochasticky, což zdůrazňovalo Csurimu fascinaci náhodou a principy nepředvídatelnosti.[17]



Obrázek 1 – Flies [18]



Obrázek 2 – Flies Transformed [18]

Práce *Flies* (1967) byla prezentována na výstavě *Coding the World* v Centre Pompidou v roce 2018 a je součástí stálé sbírky muzea Victoria and Albert Museum v Londýně.[17]

Dílo *Fly Plotter* z roku 1966 (rozměry 4"×6") bylo vystaveno v rámci první části retrospektivní výstavy *Charles Csuri Memorial Exhibition* v galerii Hopkins Hall ve městě Columbus, stát Ohio.[17] O tomto díle uvedl: „A common housefly seemed to have a place in the context of an intelligence which will replace human beings. Possibly feeding off some residue left behind when the heat of the planet burns all the circuit boards." – Charles Csuri [17]

Leonardo da Vinci Series, 1966

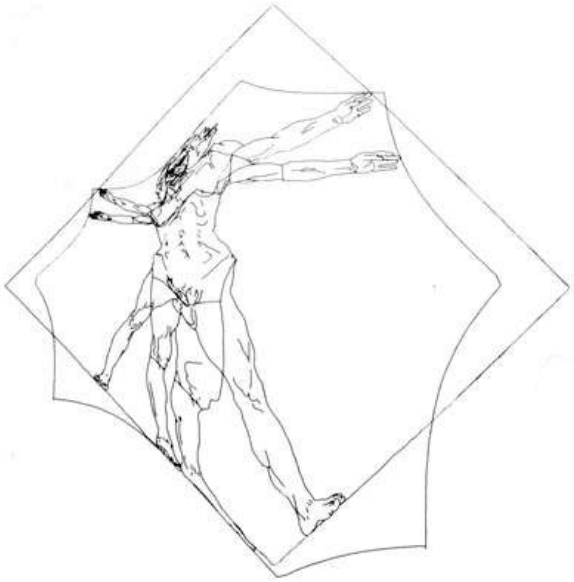
V letech 1963–1964 vytvořil Charles Csuri své první analogové počítačové práce, v nichž reinterpretoval klasické malby starých mistrů, k nimž choval hluboký obdiv. V těchto raných experimentech využíval vědecké nástroje ke zkoumání možností transformace tradičního výtvarného jazyka, čímž položil základy novému vizuálnímu stylu charakteristickému nepředvídatelností a řízeným chaosem. Skrze zjednodušování a deformaci originálních děl se Csuri soustředil na vztahy mezi pozicí, rotací a měřítkem, čímž otevřel prostor pro další digitální experimentaci.[19]

Přechod od analogových ke zcela digitálním formám mu následně umožnil vyjádřit úctu k historickým vzorům novým způsobem. Významným příkladem je reinterpretace *Vitruviánského muže* Leonarda da Vinciho, kterou Csuri realizoval s využitím digitálních technologií. Pomocí matematických operací, jako je inverze geometrických tvarů (kruhu a čtverce), transformoval ikonickou kresbu do nové vizuální podoby, v níž byl původní motiv dále modifikován prostřednictvím počítačových algoritmů. Výsledkem byla výrazná vizuální pocta, která zároveň sloužila jako demonstrace možností počítačového zpracování klasických výtvarných témat.[19]

Tato série nejenže reflektovala Csurimu úctu k da Vincimu jako umělci, ale rovněž jej vnímala jako vynálezce, vědce a vizionáře. Csuri měl k dispozici historickou publikaci o da Vincim, kterou důkladně prostudoval, a tuto inspiraci dále promítl i do názvů svých obrazových souborů – stovky z nich obsahují označení *Leo*.[19]

Série *Vitruvian Man* byla pozitivně přijata odbornou veřejností a reflektována ve významných publikacích, jako je *The Computer in Art* [20] a *Computer Graphics and Art* [21]. Práce z této série byly prezentovány na přelomové výstavě *Cybernetic Serendipity* v roce 1968, dále na retrospektivní výstavě *Beyond Boundaries* v roce 2006 v Muzeu výtvarných umění v Kaohsiungu (Tchaj-wan) a na konferenci *SIGGRAPH* v Bostonu (USA) v roce 2007.[19]

„What is meaningful to me is a world of art in the context of history. I find there are relationships between great works of art, irrespective of the time in which they were created." – Charles Csuri [19]



Obrázek 3 – From square to circle (vitruvian) [18]

Aging Process Plotter, 1967

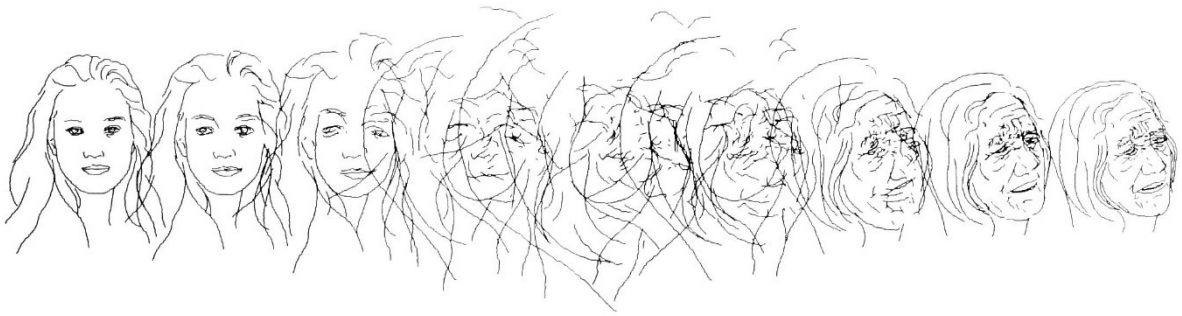
Při realizaci tohoto díla využil Charles Csuri techniku morfingu, kterou aplikoval na transformaci obličeje mladé ženy do podoby staršího člověka. Obraz byl rozčleněn do sítě úseček, na něž byla následně aplikována předem definovaná transformační pravidla. Ačkoli byl celý proces řízen výpočetním algoritmem, výsledné dílo si zachovalo prvek vizuální nečekanosti a poetiky náhody. Tento moment překvapení reflektuje Csuriho charakteristický smysl pro absurditu a hravost, který se výrazně projevuje ve zvoleném konceptu reverzního procesu stárnutí – tedy postupné proměny od starší podoby k mladší.[22]

„I try to play at the edge of reason and absurdity. It is an invitation to something alive."
– Charles Csuri [22]

Charles Csuri se ve své tvorbě intenzivně věnoval transformaci vizuálních objektů, zejména lidských tváří a zvířecích motivů. Jedna z jeho animací zahrnuje několik zásadních děl, jako jsou *Artists and a Frog*, *Hummingbird*, *Aging Process* a *Bearded Man*.[22]

Významného pokroku dosáhl Csuri při vývoji techniky morfingu – plynulé transformace jednoho obrazu v jiný, která se později stala standardem v oblasti digitální animace. Jeho inovativní přístup ovlivnil například i tvorbu animátorů ve videoklipu *Black or White* Michaela Jacksona, kde byla tato metoda využita k proměnám lidských tváří napříč pohlavími a etnickými skupinami.[22]

Pozoruhodné je, že již v roce 1967 Csuri dosahoval těchto výsledků za pomoci tehdejších technických prostředků – konkrétně pomocí děrných štítků a výpočetního systému IBM 7094. Tento stroj, jehož hodnota tehdy činila 2,9 milionu dolarů (ekvivalentně téměř 30 milionů dolarů v současnosti), byl v dané době považován za špičku dostupné výpočetní techniky.[22]



Obrázek 4 – Aging Process Plotter, ve sbírce MUDigital [22]

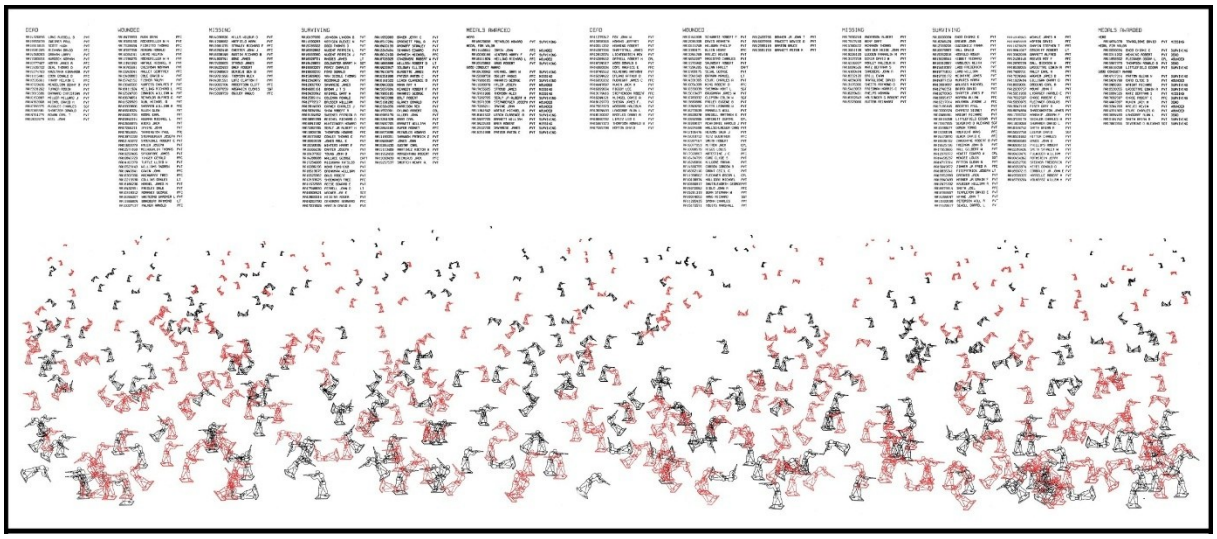
Random War, 1967

Charles Csuri vytvořil generativní systém založený na algoritmu náhodných čísel, jehož prostřednictvím vzniklo dílo *Random War*. Tento systém generoval vizuální scénáře bitev, přičemž jako výchozí datová jednotka sloužila digitalizovaná kresba vojenské figurky. Výsledná kompozice nese paralely s principy současných strategických videoher. Každému vojákovi byl automaticky přiřazen identifikační údaj – jméno, hodnost – a zároveň mu byla náhodně přisouzena role ve výsledku bitvy (např. padlý, zraněný, nezvěstný, přeživší, či vyznamenaný). Dílo tak představuje ikonický a emocionálně silný příklad raného generativního umění, v němž se snoubí výpočetní metoda s uměleckým záměrem.[23]

Random War vzniklo v období výrazného společenského napětí, kdy Spojené státy zažívaly silné vnitropolitické rozdělení v souvislosti s válkou ve Vietnamu. V té době panovala polarizace mezi zastánci technologického pokroku a těmi, kdo jej vnímali jako hrozbu vedoucí k dehumanizaci a odcizení. Pro část umělecké komunity bylo samotné využití počítače k umělecké tvorbě chápáno jako kontroverzní, či dokonce negativní čin. [23]

Toto zásadní dílo bylo poprvé prezentováno v roce 1968 na výstavě *Cybernetic Serendipity*, pořádané londýnským Institutem současného umění (ICA). Jednalo se o jednu z prvních mezinárodních výstav zaměřených na průnik technologií a umění, na níž se prezentovali nejen umělci, ale i matematici, inženýři, skladatelé a literáti. *Random War* zde zaujalo jako jedno z prvních příkladů konceptuálního umění s využitím počítačového generování a je považováno za předzvěst digitálních her. Významným způsobem přispělo k Csurimu dlouhodobému zájmu o náhodnost, herní principy a inovace v oblasti digitální tvorby.[23]

Silným momentem díla je také použití konkrétních jmen – ať již skutečných historických postav nebo kulturních ikon –které akcentují náhodnost a absurditu války. Mezi těmito jmény se objevují například Ronald Reagan, Gerald Ford, Robert F. Kennedy, Lyndon B. Johnson, W. R. Rockefeller, James Bond, Allen Kaprow, Roy Lichtenstein či George Segal. Tento aspekt díla zdůrazňuje personalizaci jinak anonymního válečného konfliktu a posiluje jeho kritický a ironický rozměr.[23]



Obrázek 5 – Random War [23]

Hummingbird a Hummingbird II, 1967

Dílo *Hummingbird* představuje významný příklad Csuriho raného zájmu o přírodní motivy a jejich počítačovou interpretaci. V této animaci dosahuje pohybu skrze postupnou deformaci objektu – kolibříka – přičemž forma připomíná dětskou hru s papírovou figurkou, kterou dítě ohýbá a otáčí ve snaze napodobit let. V tomto imaginárním scénáři kolibřík ožívá, postupně se rozpadá do jediné linie, a následně se znovu zformuje, tentokrát s opačným směrem pohybu. Počítač zde funguje jako nástroj s dokonalou pamětí, schopný generovat přesné variace. V závěrečné fázi animace se fiktivní „pozorující inteligence“ rozhodne ukončit sledování – kolibřík je postupně vymazán a obraz se navrací do prázdného prostoru.[24]

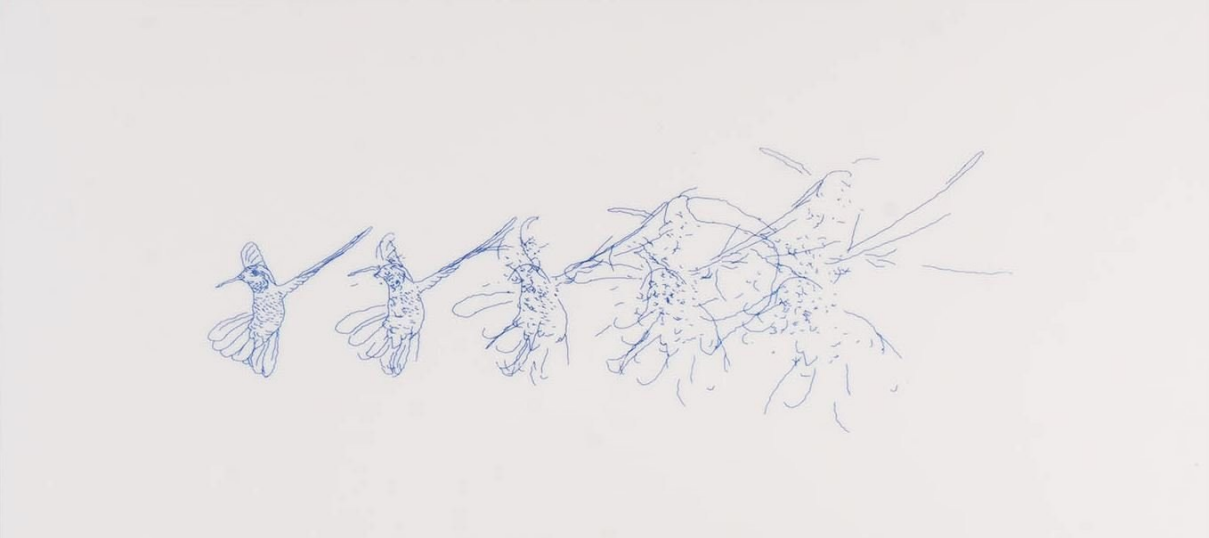
Hummingbird je považován za jedno z prvních počítačem animovaných děl Charlese Csuriho. Význam tohoto díla však nespočívá pouze v jeho technické inovativnosti, ale také v zavedení figurativní reprezentace do oblasti počítačové grafiky, která byla tehdy převážně chápána jako médium pro vizualizaci abstraktních matematických struktur. Právě kontrast a propojení mezi figurací a abstrakcí je v *Hummingbirdu* výrazným tématem – zatímco zpočátku zobrazuje rozpoznatelný tvar kolibříka, v závěru přechází do formálně abstraktní destrukce, čímž poukazuje na kompatibilitu těchto dvou vizuálních principů v rámci digitálního umění.[24]

Technicky vzato šlo o mimořádně náročný projekt. Film vznikl pomocí mikrofilmového plotru, který nakreslil více než 30 000 jednotlivých snímků přímo na filmový pás. Každý obraz byl naprogramován samostatnou děrnou kartou, což dokládá vysokou míru složitosti a časové náročnosti, jež byla typická pro rané fáze počítačové animace. Úvodní sekvence filmu zároveň slouží jako vizuální přehled technických principů a postupů, na jejichž základě bylo dílo vytvořeno – tento segment je často považován za instruktivní příklad procesů rané generativní grafiky.[24]

V roce 1968 byl *Hummingbird* zařazen do programu počítačově generovaných filmů, který uspořádalo Museum of Modern Art (MoMA) v New Yorku. Program vznikl ve spolupráci s programátorem Kenem Knowltonem, známým pro svou činnost v Bell Labs a spolupráci

s avantgardními tvůrci jako Stan VanDerBeek. Promítání proběhlo v rámci výstavy *The Machine as Seen at the End of the Mechanical Age* a zahrnovalo také filmy Johna Whitneyho. Krátce po uvedení zakoupilo MoMA *Hummingbird* do své sbírky, čímž se dílo stalo jedním z prvních počítačově vytvořených filmů akvírovaných prestižní světovou institucí.[24]

Pokračováním těchto vizuálních a konceptuálních experimentů bylo dílo *Hummingbird II*, které reflektuje Csurimu fascinaci fragmentací a vztahem mezi formou a její abstrakcí. Na rozdíl od původní animace je *Hummingbird II* jedním z mála raných děl, které existují ve fyzických podobách, např. jako fotografický výstup na plexiskle.[24]



Obrázek 6 – Hummingbird II, v soukromé sbírce [24]

„Hummingbird anticipates the future when computers will have the ability to think for themselves. The film begins with empty space and the idea the computer has an intelligence and is drawing a hummingbird. A drawing is being created with the science fiction notion of artificial intelligence and its implications for the future. An artist's hand is not making the line segments instead some unseen mysterious force is in charge. The force breaks the drawing into moving line segments which become increasingly chaotic or abstract and play with aspects of reality. Movement is achieved through distortion much like a child twisting and rotating a paper hummingbird. The child knows it cannot flap its wings but let's pretend it can. Then the hummingbird gradually collapses into a single line and reappears and moves in an opposite direction. The computer has a perfect memory and can make exact copies of everything. After several more variations the intelligence as it watches the clock, decides time is up and slowly erases the hummingbird to again become empty space.” – Charles Csuri [24]

Sine Curve Man, 1967

Dílo *Sine Curve Man* představuje zásadní mezník v rané fázi počítačového umění. Charles Csuri jej vytvořil jako autoportrét a toto dílo je považováno za první figurativní počítačovou kresbu vzniklou ve Spojených státech. V době svého působení jako profesor výtvarného umění na Státní univerzitě v Ohiu využíval Csuri k tvorbě výkonný počítač IBM 7094, který byl na počátku 60. let považován za technologickou špičku. Tento stroj sloužil mimo jiné

i NASA pro vesmírné programy Gemini a Apollo a byl nasazován v systémech protiraketové obrany.[25]

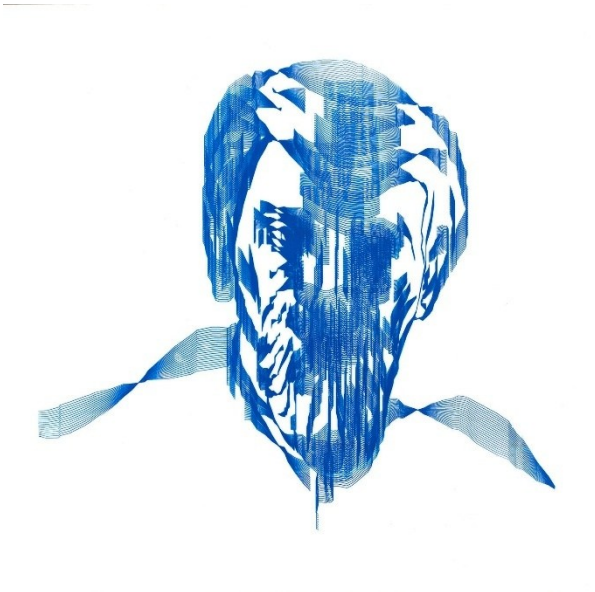
Výstupy z IBM 7094 byly ukládány na děrné štítky o velikosti 4 × 7 palců. Tyto štítky obsahovaly instrukce pro řízení bubnového plotru Cal Comp 565, který na jejich základě určoval pohyby kreslicího pera – včetně jeho zdvihu, přesunů a pokládání na papír. Výsledný obraz byl realizován jako velkoformátový výtisk na plotrovém papíře a na plexiskle, přičemž právě varianta na plexiskle je jediným exemplářem obrácené verze díla.[25]

Sine Curve Man je důležitým příkladem Csurim raného experimentování s re mediací – tedy převodem jednoho média do jiného, což je charakteristický rys digitálního věku. Toto dílo bylo zpracováno nejen jako statický obraz, ale také jako počítačová animace, čímž autor rozšiřoval hranice tradiční vizuální reprezentace směrem k multimediálním formám. V tomto kontextu je Csuri považován za jednoho z prvních, kdo do jazyka počítačové grafiky vnesl figuraci a zároveň položil základy vizuálních přechodů, které se v pozdějších dekádách staly běžnou součástí populární kultury – příkladem je videoklip *Black or White* Michaela Jacksona z roku 1991, v němž byla použita technika morfingu, jejíž základy Csuri rozvíjel již v 60. letech.[25]

Motiv *Sine Curve Man* se v Csurim díle opakovaně objevoval. Existují dvě známé verze tohoto díla, přičemž jediné velkoformátové provedení s červeným profilem bylo realizováno sítotiskem. Dílo bývá často označováno za jednoho z prvních „avatarů“ v dějinách digitálního umění. Umělecký kritik Douglas Davis jej v roce 1971 popsal jako „komplexní spojení reality a matematiky, nesoucí výraznou emocionální sílu“.[25]

Dílo vzbudilo pozornost odborné veřejnosti a bylo zařazeno kurátorkou a teoretičkou Jasiou Reichardtovou na přelomovou výstavu *Cybernetic Serendipity*, která se konala v roce 1968 v Londýnském Institutu současného umění (ICA). O rok později Reichardtová zahrnula rozbor *Sine Curve Mana* i do své vlivné publikace *The Computer in Art* (1971). Význam této práce potvrdila i akvizice její verze Whitney Museum of American Art v New Yorku, které dílo vystavilo v rámci výstavy *Programmed: Rules, Codes, and Choreographies in Art, 1965–2018* (2018–2019).[25]

V roce 1968 bylo *Sine Curve Man* zvítězil v mezinárodní soutěži počítačového umění, čímž bylo stvrzeno jeho postavení jako jednoho z nejvýznamnějších raných děl na M generativního a digitálního umění.[25]



Obrázek 7 – Sine Curve Man [25]

První tříosá frézka, která kdy byla vyrobena, 1968

V roce 1968, během svého působení na katedře strojírenství, dosáhl Charles Csuri zásadního průlomu ve své umělecké praxi. Ve spolupráci s profesorem matematiky Lesliem Millerem, členem výzkumné skupiny Computer Graphics Research Group (CGRG), využil počítačem řízenou frézku k vytvoření trojrozměrného objektu. Na základě Millerem napsaného softwarového kódu Csuri experimentoval s jednotlivými parametry až do okamžiku, kdy dosáhl požadované formy. Výsledné dílo je považováno za jednu z vůbec prvních počítačem vytvořených 3D soch a představuje významný milník nejen v rámci Csurim uměleckého vývoje, ale i v širším kontextu počítačového umění.[26]

Obrázek 8 – První 3D socha [26]

Tento experiment zásadně rozšířil Csuriho chápání možností digitálních technologií ve výtvarné tvorbě a položil základy jeho dalšího výzkumu v oblasti trojrozměrné animace a interaktivních médií, který intenzivně rozvíjel na počátku 70. let. Jeho práce v této oblasti přispěla k definování nových přístupů ke generativnímu a technologicky zprostředkovanému umění.[26]

Mirare, the Digital Art Sculpture, 2010

Dílo *Mirare* představuje monumentální instalaci o výšce přibližně 13 metrů, která inovativně propojuje fyzickou sochařskou formu s prvky digitální vizuality. Tato volně stojící struktura, svým tvarem připomínající stylizovaný černý strom, integruje devět vysokorozlišovacích obrazovek, které plní funkci „listů“. Na každé z těchto obrazovek je promítán odlišný segment kontinuální počítačově generované animace, vytvořené na míru Charlesem Csurim.[27]

Abstraktní a barevně dynamické vizuály se plynule pohybují a vytvářejí komplexní audiovizuální zážitek, který diváka podněcuje k interakci a k pozorování díla z různých perspektiv. Instalace tímto způsobem narušuje tradiční pojetí sochy jako statického objektu a redefinuje vztah mezi prostorem, pohybem a digitálním obsahem.[27]

Mirare je často označováno za jedno z prvních děl, které komplexně integrují digitální animaci do trojrozměrného prostoru. Zpochybňuje konvenční vnímání formy a zároveň otevírá nové možnosti pro uvažování o podobě a prezentaci digitálního umění ve veřejném prostoru.[27]

„No longer restricted to rectangular spaces, animation is being redefined to take on a different role.” – Charles Csuri [27]

Mirare je považováno za vůbec první digitální sochu svého druhu. Její název pochází z latinského *mirare*, což znamená „divit se“ či „obdivovat“. Toto průlomové dílo bylo dokončeno a slavnostně odhaleno 27. července 2010. Instalace jedinečným způsobem propojuje fyzickou materiální přítomnost s digitální animací, čímž vytváří nový typ multimediálního uměleckého zážitku. Dílo sestává z fyzické konstrukce, do níž je integrováno několik synchronizovaných obrazovek s vysokým rozlišením, zobrazujících vizuálně propojené segmenty dynamické abstraktní animace vytvořené přímo na míru.[27]

Barevné a tvarové prvky se na jednotlivých obrazovkách plynule transformují, čímž vzniká dojem živého, neustále se proměňujícího objektu. Animace je koncipována jako cyklický proud bez lineárního narativu, čímž je divákovi umožněno plně se ponořit do vizuální zkušenosti bez nutnosti konceptuální interpretace. Dílo lze pozorovat z různých úhlů, podobně jako tradiční sochu, čímž se vytrácí jasná hranice mezi klasickým a digitálním uměním. Začátek a konec animace splývají, což podtrhuje její kontemplativní charakter.[27]

Realizace *Mirare* byla technologicky velmi náročná a vyžadovala inovativní přístup k synchronizaci a zobrazení digitálního obsahu ve vysokém rozlišení. Dílo využívá patentovanou technologii vyvinutou společnostmi Rivet Digital a CsuriStudios, která umožňuje dokonale sladěný chod více displejů v reálném čase. Pomocí pokročilých metod digitálního kódování, ukládání a zpracování dat bylo dosaženo plynulého přehrávání animací

v mnohonásobně vyšší kvalitě, než jakou nabízí standardní technologie vysokého rozlišení.[27]

Tato platforma navíc umožňuje nasazení prakticky neomezeného počtu obrazovek, které lze rozmístit v libovolném uspořádání v rámci fyzického prostoru. Displeje pak fungují jako „okna“ do digitálního trojrozměrného prostředí umělce. Díky této flexibilitě je možné vytvářet instalace s desítkami, stovkami či tisíci obrazových jednotek, čímž se výrazně rozšiřují možnosti měřítka, kompozice i perspektivy v oblasti digitálního a interaktivního umění.[27]

Na realizaci projektu *Mirare* se podílelo množství osob a organizací, mimo jiné Christina a Jim Groteovi, Willie Grové, Caroline Csuri, Kevin Reagh, Dale McClintock, Russ Nagy, společnosti Rivet Digital, Bainter Machining and Fabricating, NuGrowth Technologies, a dále Matt Lewis a Pete Carswell.[27]

Obrázek 9 – Siluety Charlese Csuriho (napravo) povídajícího si s fanouškem (nalevo), před dílem Mirare, po slavnostním odhalení 27. července 2010 [28]

1 TEORETICKÁ ČÁST

Teoretická část této práce je zaměřena na klíčové teoretické koncepty, které jsou nezbytné pro pochopení a aplikaci pseudonáhodných generátorů čísel (PRNG) v kontextu počítačového umění, grafiky a dalších aplikací. Tato část poskytuje základní rámec pro pochopení principů fungování pseudonáhodného generování čísel, jejich využití v praxi, stejně jako související metody a techniky, jako jsou afinní transformace, které mají význam pro manipulaci s objekty v grafických aplikacích.

1.1 Pseudonáhodné generování čísel

Pseudonáhodné generování čísel je technika, která je široce využívána v mnoha oblastech počítačové vědy a inženýrství, včetně simulací, testování, herní mechaniky, kryptografie a počítačové grafiky. Tento proces se liší od skutečného náhodného generování čísel, protože číselné sekvence vytvořené pseudonáhodnými generátory čísel (PRNG) jsou deterministické, což znamená, že jsou zcela určeny počátečními hodnotami (semeny) a používaným algoritmem. Přestože čísla generovaná PRNG vypadají náhodně, mohou být předpověditelná, pokud jsou známy výchozí podmínky. To, jak PRNG fungují, jakým způsobem používají seed k inicializaci generátoru a jak se využívají v praktických aplikacích, je klíčové pro pochopení jejich roli v moderních technologiích.

1.1.1 Úvod do pseudonáhodných generátorů

Pseudonáhodné generátory čísel (PRNG) jsou algoritmy, které generují sekvence čísel, jež vypadají jako náhodné, ale ve skutečnosti jsou deterministické. Tato deterministická povaha znamená, že sekvence čísel generovaných PRNG je plně určena počáteční hodnotou (seed) a metodou generování. Taková čísla se běžně používají v aplikacích, kde je potřeba rychlé a efektivní generování "náhodných" čísel, ale kde není nezbytné, aby čísla byla zcela náhodná v pravém slova smyslu.

Pseudonáhodné generátory jsou široce využívány v počítačové grafice a digitálním umění. Například ve 3D počítačové grafice se používají k simulaci náhodných textur, barev nebo pohybů objektů. Ve hrách se PRNG používají k určení náhodnosti v herních mechanikách, jako jsou hod kostkou nebo generování náhodných událostí. Jejich výhodou je, že mohou generovat velké množství čísel rychle a efektivně, což je nezbytné pro plynulé běhy simulací nebo interaktivních aplikací.

I když se generovaná čísla z PRNG mohou zdát náhodná, nejsou skutečně náhodná. Skutečná náhoda je obtížně dosažitelná v počítačovém prostředí, protože počítače jsou deterministické systémy. To znamená, že jsou předvídatelné a mohou generovat pouze určité výsledky na základě výchozích podmínek. Na druhou stranu skutečně náhodná čísla by měla vznikat z náhodných fyzikálních procesů, jako je šum v elektronických obvodech nebo změny v atmosférických podmínkách.

Pseudonáhodné generátory čísel jsou v tomto směru kompromisem, protože i když jejich výstupy nejsou skutečně náhodné, pro většinu aplikací (jako je počítačová grafika, simulace nebo herní mechaniky) jsou dostatečně "náhodné". To znamená, že pro tyto účely mohou

PRNG efektivně nahradit skutečné náhodné generátory, které jsou obvykle nákladnější a pomalejší.

1.1.2 Princip fungování pseudonáhodných generátorů čísel

Pseudonáhodné generátory čísel začínají svou práci inicializací, která obvykle zahrnuje nastavení výchozí hodnoty, známé jako seed. Seed je klíčový prvek, protože určuje, jak bude generátor čísel začínat, a tím i celou sekvenci generovaných čísel. Vzhledem k tomu, že PRNG jsou deterministické, pokud použijeme stejný seed, vždy získáme stejnou posloupnost čísel.

Seed může být buď statický, například pevně definovaný v kódu, nebo dynamický, což znamená, že je generován na základě nějakého náhodného faktoru (například aktuálního času nebo externího zařízení, které poskytuje šum). V případě statického seedu bude každé spuštění programu, které používá stejný seed, generovat stejnou posloupnost čísel. To je výhodné pro testování nebo simulace, kde je potřeba, aby byly výsledky opakovatelné. Dynamický seed je naopak užitečný pro aplikace, kde je požadována různorodost náhodných čísel při každém běhu programu, například v hrách, kde každý nový běh generuje jiný herní zážitek.

Pseudonáhodný generátor čísel využívá určitý algoritmus k výpočtu sekvence čísel z počátečního seedu. Mezi nejběžnější algoritmy patří lineární kongruenční generátor (LCG), Mersenne Twister a algoritmy využívající bitové operace, jako je Xorshift.

1. **Lineární kongruenční generátor (LCG)**: Tento generátor používá jednoduchý rekursivní vztah k vytvoření nové hodnoty na základě předchozí. Obvyklý tvar vztahu je
   $x_{i+1} = (ax_i + c) \bmod m,$
   kde operace *mod* znamená zbytek po celočíselném dělení, x_i je aktuální hodnota, $a, c\ a\ m$ jsou konstanty (parametry generátoru), x_0 je seed.
   Tento algoritmus je efektivní, ale jeho výstupy mohou vykazovat opakující se vzory, pokud nejsou parametry správně zvoleny. LCG má omezenou periodu, což znamená, že po určitém počtu generovaných čísel začne sekvence znovu opakovat. Tento algoritmus je implementován v mnoha programovacích jazycích, včetně Javy, kde je základem pro třídu *Random*.
2. **Mersenne Twister**: Tento algoritmus je velmi oblíbený, zejména kvůli své dlouhé periodě a vysoké kvalitě náhodnosti. Mersenne Twister má periodu $2^{19937} - 1$, což znamená, že generátor bude vytvářet číselnou sekvenci po velmi dlouhou dobu, než začne opakovat hodnoty.
3. **Bitové operace (Xorshift)**: Některé moderní generátory čísel využívají bitové operace, jako je XOR (bitová operace). Tyto generátory jsou rychlé a efektivní a produkují kvalitní náhodná čísla. Algoritmus Xorshift je založen na několika jednoduchých bitových operacích, které rychle generují nová čísla.

1.1.3 Rovnoměrné spojité rozdělení

Rovnoměrné rozdělení je pravděpodobnostní model, který popisuje situaci, kdy všechny možné výsledky mají stejnou pravděpodobnost výskytu. V matematice a statistice se používá ke generování náhodných událostí nebo simulací procesů. Rozlišujeme dva typy:

Spojité rovnoměrné rozdělení se používá, když jsou možné výsledky spojité a leží v intervalu $\langle a, b\rangle$. Hustota pravděpodobnosti je konstantní a je definována jako

$$f(x) = \begin{cases} \dfrac{1}{b-a}, & \text{pro } x \in \langle a, b\rangle, \\ 0, & \text{jinak.} \end{cases}$$

Pravděpodobnost, že náhodná proměnná X spadne do podintervalu $\langle c, d\rangle$ je dána vzorcem

$$P(c \leq X \leq d) = \frac{d-c}{b-a}.$$

Pseudonáhodné generátory čísel (PRNG) často využívají rovnoměrné rozdělení jako základní mechanismus pro generování čísel. PRNG generují čísla, která vypadají náhodně, ale ve skutečnosti jsou deterministická a založená na algoritmu. Typicky vracejí čísla v intervalu $\langle 0, 1\rangle$, která jsou rovnoměrně rozložena, a tato čísla lze následně transformovat na požadovaný rozsah.

Příklad základního PRNG algoritmu:

1. Algoritmus generuje číslo u z intervalu $\langle 0, 1\rangle$.
2. Toto číslo je transformováno na požadovaný interval $\langle a, b\rangle$ vzorcem $x = a + u(b-a)$.

Rovnoměrné rozdělení je zde důležité, protože umožňuje rovnoměrné pokrytí celého intervalu, což je klíčové pro mnohé simulace a numerické metody, jako je Monte Carlo.

V Javě je třída *Random* klíčovým nástrojem pro generování náhodných čísel. Jedním z typů náhodných čísel, která může generovat, jsou čísla rovnoměrně distribuovaná v určitém intervalu. V tomto případě se zaměříme na metodu *nextDouble()*, která generuje náhodná čísla typu double v intervalu $\langle 0, 1)$, což odpovídá rovnoměrnému spojitému rozdělení. Toto rozdělení znamená, že každé číslo v tomto intervalu má stejnou pravděpodobnost výskytu. Tento proces je základem mnoha aplikací, jako jsou simulace, počítačová grafika, hry a další.

Metoda *nextDouble()* generuje náhodné číslo, které je rovnoměrně distribuováno v intervalu $\langle 0, 1)$, To znamená, že vygenerované číslo bude vždy větší nebo rovno 0 a menší než 1. Tento typ náhodného čísla je základem pro generování náhodných hodnot v mnoha oblastech, kde je potřeba rovnoměrného rozdělení hodnot v tomto intervalu.

K tomu, aby bylo možné generovat taková čísla, Java využívá pseudonáhodný generátor čísel, což znamená, že výstupy jsou deterministické, pokud je znám počáteční seed (počáteční hodnota generátoru). Tento seed určuje počáteční stav generátoru a tím i sekvenci čísel, která bude generována. Pokud je seed stejný, vždy bude generována stejná posloupnost čísel, což je užitečné pro replikovatelnost testů nebo simulací.

Třída *Random* v Javě používá Linear Congruential Generator (LCG), což je jeden z nejjednodušších a nejběžněji používaných algoritmů pro generování pseudonáhodných čísel. Tento algoritmus používá následující rekursivní vztah pro generování celých čísel

$$x_{i+1} = (ax_i + c) \bmod m,$$

kde x_i je aktuální hodnota, a je multiplikátor, c je přičítací konstanta, m je modul (modulová hodnota).

Tento generátor používá několik konstant, které jsou pečlivě navrženy tak, aby vytvářely kvalitní sekvenci pseudonáhodných čísel. Pro Java implementaci jsou konkrétní hodnoty těchto konstant, Multiplikátor (a) = 0x5DEECE66D, přičítací konstanta (c) = 0xB, modul (m) = 2^{48}.

Díky těmto parametrům má generátor periodu 2^{48}, což znamená, že po 2^{48} vygenerovaných číslech začne generátor znovu opakovat sekvenci. Toto je dostatečně dlouhá perioda pro většinu aplikací, ale pro velmi citlivé nebo specifické aplikace, jako je kryptografie, je lepší použít jiné metody generování čísel.

Po generování pseudonáhodného čísla pomocí LCG je výstup obvykle celé číslo v určitém rozsahu (například v rozmezí od 0 do $2^{48} - 1$). Aby bylo možné generovat číslo typu *double* v intervalu $\langle 0, 1)$, toto celé číslo je převedeno na reálné číslo v tomto intervalu. Proces je následující

1. Generování celého čísla: Algoritmus generuje celé číslo v rozsahu $0 \leq x_i < 2^{48}$ pomocí LCG.
2. Transformace na číslo typu double: Tato vygenerovaná hodnota je poté převedena na číslo typu double dělením maximální hodnoty generátoru $2^{48} - 1$. To znamená, že vygenerované číslo je děleno maximální možnou hodnotou, což vede k výsledku v intervalu $\langle 0, 1)$

   $$randomDouble = \frac{x_i}{2^{48} - 1}.$$

   Tímto způsobem je vygenerováno číslo typu double, které je rovnoměrně distribuováno mezi 0 a 1 (včetně 0, ale bez 1).

```
public double nextDouble() {
  return (((long)next(26) << 27) + next(27))
    / (double)(1L << 53);
}
```

Důležitým aspektem je, že číslo generované metodou nextDouble() má rovnoměrné rozdělení, což znamená, že všechna čísla v intervalu $\langle 0, 1)$ mají stejnou pravděpodobnost výskytu. Tento typ rozdělení je klíčový pro různé aplikace, jako jsou simulace, modelování, počítačová grafika nebo generování náhodných čísel v hrách, kde je třeba zajistit, aby nebyl preferován žádný konkrétní interval hodnot. Tím je zaručeno, že jakákoli hodnota mezi 0 a 1 je stejně pravděpodobná, což je základní vlastnost rovnoměrného rozdělení.

Charles Csuri, průkopník počítačového umění, často využíval matematické principy, včetně rovnoměrného rozdělení, k tvorbě svých děl. Pomocí rovnoměrného rozdělení generoval náhodné pozice a transformace objektů ve svých digitálních obrazech. Tento přístup umožnil vytvářet vizuálně zajímavé kompozice, které vypadaly organicky a nečekaně, přestože byly výsledkem algoritmických procesů.

V jeho pracích mohly být prvky rozloženy rovnoměrně po celé ploše obrazu nebo se mohly pohybovat v určitých intervalech podle generovaných pseudonáhodných hodnot. Tím dosáhl estetického efektu, který připomínal přirozené rozložení objektů v prostoru.

1.1.4 Normální rozdělení

Normální rozdělení, často označované jako Gaussovo, představuje klíčový model ve statistice, který popisuje rozložení dat kolem střední hodnoty. Typickým znakem tohoto rozdělení je symetrická křivka ve tvaru zvonu. V jejím středu leží střední hodnota, která se shoduje s mediánem i modem. Hodnoty se koncentrují kolem průměru a s rostoucí vzdáleností jejich výskyt klesá. Důležitou vlastností je, že téměř všechny hodnoty se nacházejí v rozmezí tří směrodatných odchylek od průměru. Směrodatná odchylka určuje, jak moc se data rozptylují kolem střední hodnoty.

Matematicky lze normální rozdělení vyjádřit funkcí hustoty pravděpodobnosti, která určuje pravděpodobnost, že náhodná proměnná nabude určité hodnoty. Tato funkce je definována dvěma parametry: střední hodnotou a směrodatnou odchylkou. Významné je tzv. empirické pravidlo, které říká, že přibližně 68 % hodnot leží v rozmezí jednonásobku směrodatné odchylky od průměru, 95 % maximálně ve vzdálenosti dvojnásobku směrodatné odchylky a 99,7 % maximálně ve vzdálenosti trojnásobku směrodatné odchylky od průměru.

Normální rozdělení má široké využití. V přírodních vědách modeluje například chyby měření či biologické charakteristiky, jako je výška lidí. V ekonomii a financích se používá k modelování výnosů a rizik. V pseudonáhodných generátorech čísel umožňuje generovat hodnoty, které odpovídají přirozeným jevům, často transformací z rovnoměrně rozdělených hodnot pomocí speciálních algoritmů.

Třída Random v Javě nabízí několik metod pro generování pseudonáhodných čísel, z nichž jedna je *nextGaussian(),* která generuje čísla podle normálního rozdělení (také známého jako Gaussovo rozdělení). Tento typ rozdělení je velmi běžně používaný v mnoha oblastech vědy, inženýrství a statistiky, protože mnoho reálných jevů, jako jsou měření chyb, biologické procesy, nebo distribuce některých přírodních jevů, lze modelovat pomocí normálního rozdělení. V tomto textu se podíváme na to, jak metoda *nextGaussian()* v Javě generuje čísla, která odpovídají normálnímu rozdělení, a jak tento proces funguje.

Metoda *nextGaussian()* v Javě generuje náhodné číslo podle normálního rozdělení s průměrem 0 a směrodatnou odchylkou 1. To znamená, že výsledná čísla mají střední hodnotu 0 a většina vygenerovaných hodnot bude ležet v rozmezí od -3 do 3 (přibližně 99,7 % hodnot) podle známého pravidla "68-95-99.7", které popisuje chování dat v normálním rozdělení.

Výstup této metody tedy nebude rovnoměrně distribuován (na rozdíl od metod jako *nextDouble()*), ale bude vykazovat vlastnosti, které odpovídají normálnímu rozdělení. Tato čísla mají střední hodnotu (mean) = 0 a směrodatnou odchylku (standard deviation) = 1.

Vytváření normálně distribuovaných čísel pomocí metody nextGaussian() není realizováno přímo pomocí vzorce pro normální rozdělení, ale používá se k tomu efektivní metoda, která vychází z Box-Mullerovy transformace. Tato metoda umožňuje převést dvě rovnoměrně distribuovaná náhodná čísla na dvě nezávislá normálně distribuovaná čísla.

Box-Mullerova transformace je známá metoda pro generování normálně distribuovaných náhodných čísel z rovnoměrně distribuovaných náhodných čísel. Algoritmus využívá dvě nezávislé náhodné proměnné u a v které jsou rovnoměrně distribuovány v intervalu $\langle 0, 1)$ a s je hodnota součtu čtverců obou rovnoměrně distribuovaných čísel u a v. Tyto proměnné jsou následně transformovány na dvě normálně distribuovaná čísla z_0 a z_1 pomocí následujících vzorců:

$$s = u^2 + v^2,$$

$$z_0 = \sqrt{-2 \ln u} \cdot \cos(2\pi \mathrm{v}) = \sqrt{-2 \ln s} \left(\frac{u}{\sqrt{s}}\right) = u \cdot \sqrt{\frac{-2 \ln s}{s}},$$

$$z_1 = \sqrt{-2 \ln u} \cdot \sin(2\pi \mathrm{v}) = \sqrt{-2 \ln s} \left(\frac{v}{\sqrt{s}}\right) = v \cdot \sqrt{\frac{-2 \ln s}{s}}.$$

Tato čísla z_0 a z_1 jsou nezávisle normálně distribuovaná s nulovou střední hodnotou a jednotkovou směrodatnou odchylkou. V praxi Java používá tuto metodu pro generování náhodných čísel v metodě *nextGaussian()*.

1. Generování dvou rovnoměrně distribuovaných čísel: Java generuje dvě nezávislá čísla rovnoměrně distribuovaná v intervalu $\langle 0, 1)$.
2. Aplikace Box-Mullerovy transformace: Tato čísla jsou použita pro výpočet dvou normálně distribuovaných čísel z_0 a z_1.
3. Vrácení výsledku: V metodě *nextGaussian()* je vráceno jedno z těchto normálně distribuovaných čísel (obvykle z_0).

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

public double nextGaussian() {
  if (haveNextNextGaussian) {
    haveNextNextGaussian = false;
    return nextNextGaussian;
  } else {
    double v1, v2, s;
    do {
      v1 = 2 * nextDouble() - 1;   // between -1.0 and 1.0
      v2 = 2 * nextDouble() - 1;   // between -1.0 and 1.0
      s = v1 * v1 + v2 * v2;
    } while (s >= 1 || s == 0);
    double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
    nextNextGaussian = v2 * multiplier;
    haveNextNextGaussian = true;
    return v1 * multiplier;
  }
}
```

Tento proces se opakuje pokaždé, když je metoda nextGaussian() volána.

Výsledná čísla generovaná metodou nextGaussian() mají následující vlastnosti.

- Průměr: Čísla mají průměr 0, což znamená, že se symetricky rozdělují kolem hodnoty 0.
- Směrodatná odchylka: Čísla mají směrodatnou odchylku 1, což znamená že přibližně 99,7 % hodnot bude ležet v rozmezí od -3 do 3.
- Výskyt: Vygenerovaná čísla jsou rozdělena podle normálního rozdělení, což znamená, že většina hodnot bude vyskytovat v blízkosti průměru, zatímco hodnoty daleko od průměru (v oblastech 3 nebo více směrodatných odchylek) jsou méně pravděpodobné.

Tento typ rozdělení je důležitý v mnoha oblastech aplikované statistiky, jako je analýza dat, simulace a strojové učení, protože mnoho přírodních procesů a měření se přirozeně přibližuje normálnímu rozdělení.

Charles Csuri využíval normální rozdělení k vytváření variací ve svých dílech, čímž dosahoval realistického a organického vzhledu kompozic. Místo striktně rovnoměrných prvků umožňovalo normální rozdělení generovat vizuální prvky, které se více přibližují přirozeným jevům – například hustotě částic, variacím v barvách nebo tvaru objektů. To přispívalo k bohatším a dynamičtějším obrazům, které působily přirozeně a harmonicky.

1.1.5 Pseudonáhodné generátory ve Fortranu

V době, kdy Charles Csuri začal experimentovat s počítačovou grafikou a digitálním uměním, byl Fortran jedním z hlavních programovacích jazyků používaných ve vědeckých výpočtech a inženýrských aplikacích. Využíval se zejména pro simulace a numerické metody, kde bylo nezbytné generovat náhodná čísla pro různé typy výpočtů, například v simulacích fyzikálních

procesů nebo při analýze dat. Pseudonáhodné generátory čísel (PRNG) v této době byly nezbytné pro generování náhodných hodnot, ale implementace v jazyce Fortran v té době nebyly tak sofistikované jako dnes.

Charles Csuri začal používat Fortran v polovině 60. let, kdy byl jazyk stále v počátcích svého rozvoje. V té době se počítače používaly především pro vědecké výpočty a Fortran byl jedním z nejvíce rozšířených jazyků pro tuto oblast. Fortran byl v té době oblíbený nejen pro své rychlé výpočty, ale také pro svou schopnost pracovat s matematickými funkcemi, což zahrnovalo generování náhodných čísel pro simulace a analýzy.

V době, kdy Csuri začal pracovat s Fortranem, nebyly k dispozici žádné specializované knihovny nebo funkcionality pro generování náhodných čísel v takové podobě, jak je známe dnes. Generování náhodných čísel v těchto raných verzích Fortranu bylo buď implementováno přímo v programovacím jazyce pomocí základních aritmetických operací, nebo bylo realizováno pomocí externích knihoven napsaných uživateli.

Ve Fortranu 66, který byl jednou z prvních verzí jazyka Fortran, nebyly přímo k dispozici funkce pro generování náhodných čísel. To znamenalo, že programátoři, včetně Csuriho týmu, museli implementovat vlastní generátory pseudonáhodných čísel. Typicky byly používány lineární kongruenční generátory (LCG), které byly jednoduché na implementaci a rychlé na výpočty. Ve Fortranu 77, který přišel později, už byly k dispozici vestavěné funkce pro generování náhodných čísel.

V průběhu doby, kdy Csuri pracoval na svých projektech s digitálním uměním, se pseudonáhodné generátory čísel používaly nejen pro vědecké výpočty, ale také pro generování vizuálních prvků ve formě náhodně umístěných objektů nebo pro výpočty náhodných hodnot v simulacích. Pseudonáhodná čísla byla klíčová pro generování "náhodných" tvarů, pohybů nebo barev v počítačové grafice.

V době, kdy Fortran 66 dominoval, nebyla k dispozici žádná vestavěná podpora pro generování náhodných čísel. Programátoři se tedy museli obrátit na vlastní implementace pseudonáhodných generátorů. Nejčastěji používaný algoritmus v té době byl lineární kongruenční generátor (LCG), jak jsme již dříve zmínili. Tento generátor byl implementován v základním jazyce Fortran pomocí aritmetických operací, kde generování náhodného čísla probíhalo v několika krocích:

1. Inicializace seed: Programátor si určil počáteční hodnotu, tzv. seed, která byla použitá jako počáteční hodnota pro generování sekvence.
2. Výpočet nového čísla: Na základě seed a několika parametrů generátoru (multiplikátor, přičítací konstanta a modul) byl vypočítán nový výstup, který představoval pseudonáhodné číslo.
3. Opakování procesu: Tento proces se opakoval pro každé nové číslo, které generátor vygeneroval.

Příklad implementace generátoru náhodných čísel ve Fortranu 66 může vypadat takto:

```
INTEGER :: seed
REAL :: random_number
seed = 12345  ! Inicializace seedu
random_number = MOD(seed * 1664525 + 1013904223, 2**32) / 2**32
PRINT *, random_number  ! Vytiskne náhodné číslo mezi 0 a 1
```

Tento jednoduchý generátor používá základní lineární kongruenční generátor (LCG), kde je seed násoben určitým číslem, přičtena konstanta a výsledek je omezen modulo 2^{32}.

S příchodem Fortranu 77 se situace zlepšila a programátoři již nemuseli implementovat vlastní pseudonáhodné generátory. Fortran 77 začal obsahovat vestavěné funkce pro generování náhodných čísel, jako je funkce RANDOM_NUMBER, která umožnila snadno generovat náhodná čísla rovnoměrně distribuovaná v intervalu $\langle 0, 1)$.

Příklad použití RANDOM_NUMBER ve Fortranu 77:

```
REAL :: x
CALL RANDOM_NUMBER(x)
PRINT *, x  ! Vytiskne náhodné číslo mezi 0 a 1
```

Tato funkce byla snadná na použití a umožnila Csuriho týmu generovat náhodná čísla bez nutnosti implementovat vlastní algoritmus. Za tímto účelem byla funkce RANDOM_NUMBER postavena na metodách, jako je linear congruential generator, ale s pečlivě vybranými parametry, které minimalizovaly opakování sekvencí a zlepšily statistické vlastnosti generovaných čísel.

Pro větší kontrolu nad generováním náhodných čísel bylo možné použít funkci RANDOM_SEED, která umožnila inicializaci generátoru náhodných čísel pomocí určitého seedu. Tato schopnost byla důležitá pro replikovatelnost experimentů a testování, což bylo zásadní pro vědecké výpočty a pro Csuriho tým při práci na uměleckých projektech, kde byla potřeba opakovat určité procesy s identickými náhodnými hodnotami.

Příklad použití RANDOM_SEED ve Fortranu 77:

```
INTEGER :: seed(1)
REAL :: x
seed(1) = 12345
CALL RANDOM_SEED(put=seed)
CALL RANDOM_NUMBER(x)
PRINT *, x  ! Vytiskne náhodné číslo mezi 0 a 1
```

Díky této funkci mohl Csuri specifikovat seed a získat stejnou sekvenci náhodných čísel při každém spuštění programu, což bylo užitečné pro generování konzistentních výsledků.

Pseudonáhodné generátory čísel ve Fortranu byly jednoduché na implementaci, což bylo výhodné v době omezených výpočetních prostředků. Nicméně, tyto generátory měly několik omezení, jako je krátká perioda a možnost vzorování, což znamenalo, že po určitém počtu generovaných čísel začínala sekvence opakovat. To bylo nevhodné pro dlouhodobé simulace nebo aplikace vyžadující silnější náhodnost. Generátory používané ve Fortranu 77 byly stále založeny na lineárním kongruenčním generátoru, ale jejich statistické vlastnosti byly vylepšeny pomocí pečlivě vybraných parametrů, což zlepšovalo kvalitu náhodnosti.

Přestože byly tyto generátory v té době velmi užitečné a efektivní pro generování náhodných hodnot pro simulace a umělecké projekty, dnes existují pokročilejší metody pro generování náhodných čísel, které nabízejí lepší statistické vlastnosti a delší periodu. Moderní algoritmy, jako je Mersenne Twister, mají výrazně delší periodu a poskytují lepší náhodnost, ale pro většinu aplikací v době, kdy Csuri pracoval, byly generátory jako LCG dostačující pro jeho potřeby.

1.1.6 Použití pseudonáhodných generátorů v umělecké tvorbě

V době, kdy Charles Csuri začal pracovat s počítačovou grafikou a digitálním uměním, byly možnosti pro generování náhodných čísel v počítačové grafice omezené. Přesto díky využití pseudonáhodných generátorů čísel (PRNG) mohl Csuri vytvořit vizuální prvky, které vypadaly náhodně, ale byly ve skutečnosti plně deterministické, což znamenalo, že i když výsledky vypadaly jako náhoda, byly řízeny algoritmickými procesy. Tento přístup umožnil využívat matematiku a výpočetní techniku k vytvoření vizuálních prvků, které se chovaly „náhodně“, ale byly ve skutečnosti výsledkem přesně definovaných výpočtů.

Pseudonáhodné generátory čísel poskytly Csuriho týmu nástroj pro generování grafických elementů, jejichž rozmístění nebo pohyb vypadaly přirozeně, ale byly generovány na základě algoritmů, které byly plně pod kontrolou umělce. Tento nástroj umožnil například generovat náhodně umístěné body, tvary nebo objekty na obrazovce, což bylo klíčové pro dosažení estetického efektu náhody v jeho digitálních obrazech a animacích. Jedním z nejběžnějších způsobů, jakým Csuri využíval pseudonáhodné generátory čísel, bylo generování náhodných poloh bodů nebo tvarů v grafickém prostoru. Tato metoda zahrnovala generování náhodných souřadnic x a y, které byly následně použity k určení polohy bodu nebo tvaru na obrazovce. Generování těchto náhodných hodnot bylo realizováno pomocí generátorů, které vytvářely čísla v rozsahu od 0 do 1. Tato čísla byla následně přepočítána na souřadnice v daném rozsahu (například na velikost obrazovky nebo plátna), což vedlo k tomu, že objekty byly rozmístěny náhodně po obrazovce.

Pro generování tvarů, jako jsou kruhy nebo čtverce, mohl Csuri využívat stejný princip: pseudonáhodně generovat pozici, velikost, a případně i barvu těchto tvarů. Tento přístup umožnil vytvářet obrazy, ve kterých objekty nebyly pravidelně nebo symetricky rozmístěné, ale měly zdání náhody. Algoritmus mohl například rozhodnout, zda bude tvar malý nebo velký, a následně generovat jeho polohu na základě náhodně vygenerovaných čísel. Tento typ generování je základním kamenem mnoha moderních technik počítačové grafiky, kde je

potřeba vytvářet přirozeně vypadající objekty nebo textury, které nejsou monotónní nebo pravidelně uspořádané.

Kromě toho, pseudonáhodné generátory čísel umožnily Csuriho týmu generovat textury, které vypadaly přirozeně, ale byly vygenerovány na základě matematických algoritmů. Tento přístup vedl k vytváření vizuálních efektů, jako je například šum nebo jiné náhodné vzory, které byly použity k simulaci textur na 3D objektech nebo k vytvoření povrchů, které vypadaly organicky. Tato metoda generování textur, která využívala náhodná čísla k vytvoření různých vizuálních efektů, byla zásadní pro dosažení realismu a komplexity v Csuriho dílech.

Jedním z prvních významných děl, kde Csuri využil pseudonáhodné generátory čísel k dosažení náhodného vzhledu, bylo dílo "Flies" (Mouchy), vytvořené v roce 1966. Toto dílo představovalo statický obraz, kde byly mouchy rozmístěny pomocí náhodně generovaných souřadnic. Csuri použil generátory čísel k tomu, aby na plátně náhodně rozmístil jednotlivé mouchy, což dalo výslednému obrazu efekt, jako by objekty vykazovaly „náhodné“ umístění a distribuci. Tento efekt byl dosažen prostřednictvím algoritmu, který na základě pseudonáhodných čísel generoval pozice pro jednotlivé mouchy, čímž se vytvořil dojem náhody.

Podobně byl využit generativní přístup i v dalším projektu "Random War of Soldiers" (1968), který také vytvořil statický obraz, ale v tomto případě s vojáky, kteří byli rozmístěni a vykazovali různé náhodné vlastnosti, jako je směr pohybu nebo jejich stav. I zde se Csuri rozhodl použít pseudonáhodné generátory čísel k určení rozmístění vojáků na plátně, což vedlo k vytvoření komplexního a vizuálně náhodného vzhledu. Výsledný obraz naznačoval chaos a pohyb, ale vše bylo deterministické a plně pod kontrolou generativního algoritmu, což poskytovalo umělci opakovatelnost a kontrolu nad generovaným výsledkem.

Tento způsob využívání pseudonáhodných generátorů čísel k vytvoření vizuálních efektů náhody a chaosu měl velký význam pro vývoj počítačového umění, protože umožnil umělcům integrovat matematické principy do jejich tvořivých procesů. Tímto způsobem Csuri vytvářel interaktivní a vizuálně dynamické obrazy, které zůstaly pod jeho kontrolou, ale přitom vypadaly „náhodně“, což bylo pro jeho práci charakteristické. Dnes jsou podobné techniky stále základním nástrojem pro generování realistických simulací a efektů v počítačové grafice, animacích a dalších oblastech digitálního umění.

1.2 Afinní transformace

Afinní transformace jsou matematické operace, které transformují objekty v geometrickém prostoru, přičemž zachovávají určité základní geometrické vlastnosti, jako je paralelnost přímek a poměr vzdáleností mezi body na přímce. Tyto transformace jsou velmi důležité v různých oborech, včetně počítačové grafiky, robotiky a zpracování obrazu, kde se používají k manipulaci s objekty a obrazy bez narušení jejich základní struktury.

Afinní transformace se vyznačují několika klíčovými vlastnostmi. Jednou z nich je zachování paralelnosti: přímky, které byly paralelní před transformací, zůstávají paralelní i po ní. Dále afinní transformace zachovávají poměr délek úseček na stejné přímce, což znamená, že poměr

vzdáleností mezi body na přímce zůstává nezměněn. Kromě toho jsou afinní transformace lineární, což znamená, že mohou být vyjádřeny jako kombinace lineární transformace a translace. Tato lineární část zahrnuje operace jako rotaci, škálování a zkosení (shearing). Afinní transformace také zachovávají kolinearitu bodů, což znamená, že body, které byly na jedné přímce před transformací, na ní zůstanou i po ní.

V matematice se afinní transformace obvykle vyjadřuje pomocí matice a vektorů. Pokud máme bod x v prostoru, může být afinní transformace vyjádřena rovnicí

$$\boldsymbol{x'} \rightarrow \boldsymbol{A}x + \boldsymbol{b}.$$

V této rovnici **x** reprezentuje původní vektor bodu, **x′** je transformovaný vektor bodu, **A** je matice, která popisuje lineární část transformace, a **b** je vektor, který popisuje translaci.

Afinní transformace zahrnují několik základních typů operací. **Translace (Posunutí)** přesouvá objekt v prostoru bez změny jeho tvaru nebo orientace. **Rotace (Otočení)** mění orientaci objektu kolem pevného bodu nebo osy. **Škálování (Změna měřítka)** zvětšuje nebo zmenšuje objekt a může být izotropní (stejné ve všech směrech) nebo anizotropní (různé v různých směrech). **Shearing (Zkosení)** deformuje objekt tak, že se jeho tvar mění, zatímco jeden rozměr se posune úměrně jinému. **Reflexe (Zrcadlení)** převádí body objektu podle osy symetrie, což vede k zrcadlovému obrazu objektu.

Příklad afinní transformace ve dvourozměrném prostoru lze uvést pomocí bodu [x, y] na rovině. Obecná afinní transformace tohoto bodu může být vyjádřena jako

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}.$$

Zde prvky matice a_{11}, a_{12}, a_{21}, a_{22} odpovídají za škálování, rotaci a zkosení, zatímco prvky t_x, $t_y$reprezentují translaci.

Afinní transformace tak umožňují širokou škálu operací, které lze kombinovat a použít k transformaci objektů v prostoru při zachování určitých geometrických vlastností.

1.2.1 Translace

Translace neboli posunutí, je jednou z nejzákladnějších a nejčastěji používaných operací v rámci afinních transformací.

Translace je typ geometrické transformace, která se používá k přesunu objektů v prostoru bez změny jejich tvaru, velikosti nebo orientace. V nejjednodušší podobě se jedná o proces, kdy se každý bod objektu posune o stejnou vzdálenost ve stejném směru. To znamená, že translace zachovává všechny původní geometrické vztahy mezi body objektu, pouze mění jejich absolutní polohu v prostoru.

Translace se od ostatních afinních transformací, jako jsou rotace, škálování nebo zkosení, liší tím, že nemění tvar nebo velikost objektu, ale pouze jeho polohu. Díky tomu je translace snadno pochopitelná a vizualizovatelná, což z ní činí důležitý nástroj v různých oborech, včetně počítačové grafiky, geometrie a fyziky.

V matematice je translace formálně definována jako operace, která k původním souřadnicím bodu přičítá konstantní vektor. Tento vektor, nazývaný také translační vektor, určuje směr a velikost posunutí. Pokud máme bod P s počátečními souřadnicemi [x, y] v dvourozměrném prostoru, po aplikaci translace, kterou reprezentuje vektor $\mathbf{t} = (t_x, t_y)$, získáme nový bod P′ se souřadnicemi [x', y'], které lze vyjádřit jako

$$x' = x + t_x,$$
$$y' = y + t_y.$$

Podobně v trojrozměrném prostoru se translace bodu $\boldsymbol{P}[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ vyjádří rovnicemi:

$$x' = x + t_x,$$
$$y' = y + t_y,$$
$$z' = z + t_z,$$

kde t_x, t_y a t_z jsou složky translačního vektoru v osách x, y a z.

V rámci afinních transformací lze translaci rovněž vyjádřit pomocí matic. Ve dvourozměrném prostoru se k tomu využívají homogenní souřadnice, kde je každému bodu přiřazena další souřadnice (obvykle 1), což umožňuje reprezentovat translace jako součást obecné matice afinních transformací. Matice translace pro dvourozměrný prostor vypadá následovně:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Pro trojrozměrný prostor je matice translace:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Použití homogenních souřadnic je klíčové, protože umožňuje, aby všechny afinní transformace (včetně translací, rotací, škálování a zkosení) byly sjednoceny do jediného matematického rámce, který lze efektivně zpracovávat pomocí maticových operací.

Translace nachází široké uplatnění v různých oblastech. V počítačové grafice se používá například při animacích, kde objekty musí být přesouvány z jedné pozice na druhou. Při modelování reálného světa v trojrozměrném prostoru je translace nezbytná pro správné umístění objektů do scény.

Ve zpracování obrazu může translace sloužit k posunu obrazu v rámci souřadnicového systému, což je běžné například při korekci chyb způsobených chvěním kamery nebo při vyhledávání objektů v obraze.

V robotice je translace klíčová pro navigaci a pohyb robotů, zejména při plánování trajektorií, kde je nutné přesně určit, jak se robot bude pohybovat v prostoru, aniž by došlo ke změně orientace jeho těla.

V digitálním umění, zejména v oblasti počítačově generovaného umění, se translace využívá k vytváření a manipulaci obrazů a objektů v digitálním prostoru. Jedním z průkopníků v této oblasti byl Charles Csuri, který je často označován za „otce digitálního umění“ a který zásadně přispěl k rozvoji počítačově generovaných obrazů.

Csuri využíval translaci a další afinní transformace k vytváření komplexních vizuálních kompozic. Ve svých pracích kombinoval jednoduché geometrické tvary, jako jsou linie a body, které pak pomocí translací přesouval a skládal do komplexnějších struktur. Translací a jinými transformacemi, jako je rotace a škálování, dosahoval dynamických efektů, které byly pro digitální umění v 60. a 70. letech průkopnické.

Csuriho práce ukázaly, jak mohou být matematické transformace, jako je translace, použity k vytvoření abstraktního umění, které by bylo obtížné nebo nemožné vytvořit tradičními výtvarnými metodami. Jeho využití translací ukazuje, jak mohou matematické koncepty najít uplatnění nejen v technických oborech, ale i v umění, kde přinášejí nové možnosti pro kreativní vyjádření.

1.2.2 Rotace

Rotace je specifickým typem afinní transformace, která posouvá každý bod objektu po kruhové dráze kolem pevného bodu nebo osy. Tento pohyb zachovává velikosti a úhly, což činí rotaci izometrickou transformací. V afinní soustavě jsou rotace reprezentovány maticemi, které se aplikují na vektory bodů, aby došlo k jejich otáčení.

V rovině (2D) lze rotaci o úhel θ kolem počátku vyjádřit maticí

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Pokud aplikujeme tuto matici na bod $[x, y]$, získáme nový bod $[x', y']$ podle vzorce

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}.$$

V trojrozměrném prostoru (3D) je rotace složitější, protože může probíhat kolem libovolné osy. Nejčastěji se používají matice rotací kolem os x, y a z

- kolem osy x:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix},$$

- kolem osy y:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix},$$

- kolem osy z:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

2 PRAKTICKÁ ČÁST

2.1 Specifikace a požadavky

Tato kapitola definuje obecné požadavky na simulační aplikaci, která umožňuje vizuální tvorbu obrazů inspirovaných uměleckým stylem Charlese Csuriho. Aplikace se skládá z několika samostatných simulačních scén, které využívají náhodnost, geometrické transformace a variabilní datové vstupy k vytvoření generativních kompozic.

Funkční požadavky definují, co má aplikace dělat – tedy konkrétní funkce, akce a chování systému z pohledu uživatele (např. generování dat, zobrazení výsledků nebo reakce na vstup). Naproti tomu nefunkční požadavky určují, jak má být aplikace navržena a jak kvalitně má fungovat – tedy její výkonnost, použitelnost, spolehlivost, přenositelnost či rozšiřitelnost.

2.1.1 Funkční požadavky

- **Spuštění a ovládání**
    - Po spuštění aplikace je uživateli nabídnuto rozhraní pro výběr jednotlivých simulačních scén. Každá scéna se otevírá v samostatném okně a pracuje nezávisle na ostatních.
    - Uživatel má možnost zadat parametry simulace, spustit výpočet a zobrazit výsledek v grafické podobě.
- **Náhodné generování dat**
    - Aplikace umožňuje generování obrazových prvků pomocí pseudonáhodných procesů.
    - K dispozici je možnost zadat počáteční hodnotu (tzv. „seed“), což umožňuje opakovatelnost výstupu.
    - Různé části aplikace mohou využívat různé typy rozdělení náhodných hodnot (např. rovnoměrné, normální) v závislosti na účelu konkrétní simulace.
- **Grafické zobrazení výsledků**
    - Každá simulační scéna má vlastní výstupní oblast, kde jsou vizualizovány vygenerované prvky.
    - Výstup je škálovatelný a automaticky se přizpůsobuje velikosti okna, čímž se zachovává použitelnost na různých rozlišeních.
    - Uživatel může vidět originální i transformovanou podobu obrazu, případně porovnat různé stavy dat v rámci jedné scény.
- **Uživatelská interakce**
    - Aplikace umožňuje uživatelské zadání číselných a výběrových vstupů, kterými lze ovlivnit generování, zobrazení nebo transformaci objektů.

    - Ovládací prvky zahrnují mimo jiné vstupní pole, přepínače, posuvníky nebo rozbalovací nabídky.
    - Výsledky simulace jsou generovány na základě těchto vstupů, přičemž změna hodnot vede ke změně zobrazeného výstupu.
- **Interaktivita a odezva**
    - Aplikace obsahuje základní interaktivní prvky, jako je zvýraznění objektů po najetí myší nebo vizuální propojení souvisejících prvků.
    - V případě chybného vstupu je uživatel upozorněn prostřednictvím informačního dialogu.
    - Aplikace umožňuje okamžité překreslení scén na základě změny parametrů nebo rozměrů okna.

2.1.2 Požadavky na uživatelské rozhraní

- Rozhraní musí být přehledné, intuitivní a rozdělené podle jednotlivých simulačních scén.
- Vstupní hodnoty musí být validovány tak, aby se zabránilo chybám při generování výstupu.
- Uživatelské rozhraní musí poskytovat jednoznačnou vizuální vazbu mezi vstupními parametry a výstupem simulace.
- Každá simulační scéna by měla být jasně vizuálně oddělená a poskytovat dostatek prostoru pro zobrazení výstupu.

2.1.3 Nefunkční požadavky

- **Přenositelnost**
    - Aplikace musí být schopna běžet na všech běžných desktopových operačních systémech podporujících vybranou verzi Javy a JavaFX.
- **Výkon**
    - Při běžném rozsahu dat by aplikace měla poskytovat plynulé vykreslování bez znatelného zpoždění.
    - Výpočetně náročné operace (např. generování většího množství prvků) by měly být optimalizovány.
- **Modularita a rozšiřitelnost**
    - Struktura aplikace musí umožňovat snadné přidání nových simulačních scén bez nutnosti zásahu do existujících modulů.
    - Všechny scény jsou navrženy jako nezávislé komponenty s vlastním uživatelským rozhraním a řídicí logikou.

- **Robustnost a validace**
    - Aplikace musí být schopna zachytit a korektně zpracovat chybné vstupy, přičemž nesmí dojít k jejímu zhroucení.
    - Uživatelé musí být srozumitelně informováni o chybách a možnostech jejich opravy.
- **Uživatelský komfort**
    - Rozhraní musí být přizpůsobeno běžným zvyklostem uživatelů, včetně vizuálních zpětných vazeb a snadné manipulace s parametry.
    - Je doporučeno využívat běžné ovládací prvky a poskytovat okamžitou odezvu na vstupy uživatele.

2.2 Architektura aplikace a použité technologie

Aplikace je navržena s důrazem na modularitu a přehlednost. Je rozdělena do samostatných částí, které zajišťují jednotlivé aspekty jejího chodu – uživatelské rozhraní, řídicí logiku a datové struktury. Každá simulační scéna je implementována jako oddělený modul s vlastním rozhraním a obslužnou logikou. Díky tomu je možné jednotlivé části aplikace vyvíjet, ladit a případně rozšiřovat nezávisle.

Uživatelské rozhraní každé scény je definováno pomocí externího souboru ve formátu FXML, který umožňuje oddělení vizuální struktury od aplikační logiky. Interaktivní chování a reakce na vstupy uživatele jsou zajištěny prostřednictvím řídicích tříd, které propojují zobrazení s výpočetními funkcemi. Výpočetní a datová logika je u složitějších částí oddělena do samostatných metod nebo objektů, zatímco u jednodušších scén je integrována přímo v řídicím kódu. Tento přístup odpovídá rozsahu projektu a přispívá k celkové přehlednosti.

Každý modul zajišťuje samostatnou vizualizaci dat založenou na náhodných procesech a umožňuje uživateli upravit parametry simulace. Díky tomu může být aplikace dále rozšiřována o další scénáře bez nutnosti zásahu do existujícího kódu.

2.2.1 Programovací jazyk – Java

Aplikace je implementována v programovacím jazyce Java, který je široce využíván pro vývoj multiplatformních aplikací. Java je objektově orientovaný jazyk, což podporuje modulární a přehlednou strukturu kódu. Díky principu "napiš jednou, spusť kdekoli" (Write Once, Run Anywhere) je možné aplikaci provozovat na různých operačních systémech bez nutnosti úprav zdrojového kódu.[29] Silný typový systém a správa paměti prostřednictvím automatického odstraňování nepoužívaných objektů přispívají k robustnosti a bezpečnosti aplikace.[30]

2.2.2 Grafický framework – JavaFX

Pro tvorbu grafického uživatelského rozhraní a vizualizaci výstupů aplikace byl použit framework JavaFX. JavaFX umožňuje vytvářet moderní a interaktivní desktopové aplikace

s podporou 2D grafiky, animací a efektů. Díky své architektuře umožňuje oddělení vizuální vrstvy od aplikační logiky, což přispívá k lepší udržovatelnosti a rozšiřitelnosti kódu.[31]

2.2.3 Deklarativní návrh rozhraní – FXML

Struktura uživatelského rozhraní je definována pomocí FXML, což je XML-založený jazyk umožňující deklarativní popis rozložení komponent. Tento přístup umožňuje oddělit vzhled aplikace od její logiky, což usnadňuje vývoj a údržbu. FXML soubory lze načítat za běhu aplikace, což umožňuje dynamické změny uživatelského rozhraní bez nutnosti rekompilace.[31]

2.2.4 Nástroj pro návrh rozhraní – Scene Builder

Pro vizuální návrh uživatelského rozhraní byl využit nástroj Scene Builder, který umožňuje sestavovat FXML soubory prostřednictvím grafického rozhraní. Uživatelé mohou přetahovat komponenty do pracovního prostoru, nastavovat jejich vlastnosti a automaticky generovat odpovídající FXML kód. Tento nástroj zjednodušuje a urychluje proces návrhu uživatelského rozhraní, zejména pro vývojáře bez hlubších znalostí FXML.[32]

2.2.5 Standardní knihovny jazyka Java

Aplikace využívá standardní knihovny jazyka Java pro různé účely, jako je práce s kolekcemi, vstupně-výstupní operace, zpracování textu a generování náhodných hodnot. Tyto knihovny poskytují širokou škálu funkcionalit, které usnadňují vývoj a zvyšují efektivitu kódu. Například balíčky pro práci s kolekcemi umožňují efektivní správu datových struktur, zatímco knihovny pro vstupně-výstupní operace usnadňují práci se soubory a datovými proudy.[30]

2.2.6 Práce se soubory CSV

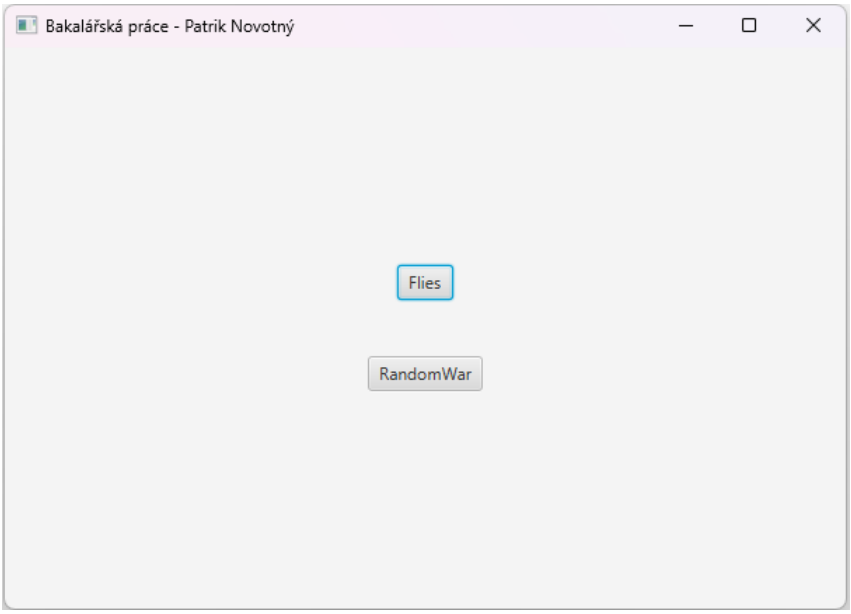
Pro načítání a zpracování vstupních dat aplikace využívá soubory ve formátu **CSV** (Comma-Separated Values). Tento formát je široce používaný pro ukládání tabulkových dat a je podporován mnoha aplikacemi, včetně tabulkových procesorů. CSV soubory umožňují snadnou výměnu dat mezi různými systémy a aplikacemi.

2.2.7 Vývojové prostředí – IntelliJ IDEA

Pro vývoj aplikace bylo použito integrované vývojové prostředí IntelliJ IDEA, které poskytuje širokou škálu nástrojů pro efektivní vývoj v jazyce Java. Mezi klíčové funkce patří inteligentní doplňování kódu, refaktoring, integrovaný debugger a podpora pro práci s verzovacími systémy. IntelliJ IDEA také umožňuje snadnou integraci s nástroji pro návrh uživatelského rozhraní, jako je Scene Builder, což usnadňuje vývoj grafických aplikací.[33]

2.3 Uživatelské rozhraní a implementace obrazů

Aplikace vytvořená v rámci této bakalářské práce je zahajována jednoduchým startovacím oknem, které slouží jako vstupní bod k výběru konkrétní vizualizace. Uživatel má na výběr mezi dvěma samostatnými moduly –*Flies* a *RandomWar* – které odpovídají dvěma odlišným generativním dílům Charlese Csuriho. Rozhraní je záměrně minimalistické a přehledné, aby nenarušovalo pozornost od hlavního obsahu aplikace. Po kliknutí na jedno z tlačítek je uživatel přesměrován do příslušné části, kde může zadávat parametry a spustit samotné generování obrazů. Toto úvodní okno tak plní funkci jednoduchého, ale efektivního rozcestníku mezi jednotlivými vizualizacemi.



Obrázek 10 – Vstupní okno aplikace

2.3.1 Flies a Flies Transformed

Analýza originálního díla:

Dílo *Flies*, vytvořené Charlesem Csurim v roce 1966, patří mezi ikonické příklady raného generativního umění a představuje kombinaci náhodnosti, algoritmického myšlení a výtvarné formy. Na první pohled působí jako jednoduchá kompozice tvořená stovkami malých mušek rozptýlených v prostoru, při bližším zkoumání je však zřejmé, že jde o výsledek přesně řízeného výpočetního procesu, který stojí na propojení programování, matematiky a estetiky.

Při tvorbě obrazu byl nejprve definován prostor, do kterého byly mušky rozmístěny. Tento prostor měl podobu série soustředných kruhů o přesném průměru jednoho palce. Ačkoliv tyto kruhy nejsou ve výsledné kompozici vizuálně zobrazeny, tvoří základní strukturu, podle níž byly objekty generovány. Každý kruh představoval zónu, v níž mohly být náhodně umístěny jednotlivé mušky. K určení jejich přesných pozic byl použit pseudonáhodný generátor čísel, který vybíral souřadnice v polárním souřadném systému. Výsledné hodnoty byly následně

převedeny na kartézské souřadnice, čímž bylo dosaženo rovnoměrného, avšak vizuálně přirozeného rozložení prvků po celém obrazovém poli.

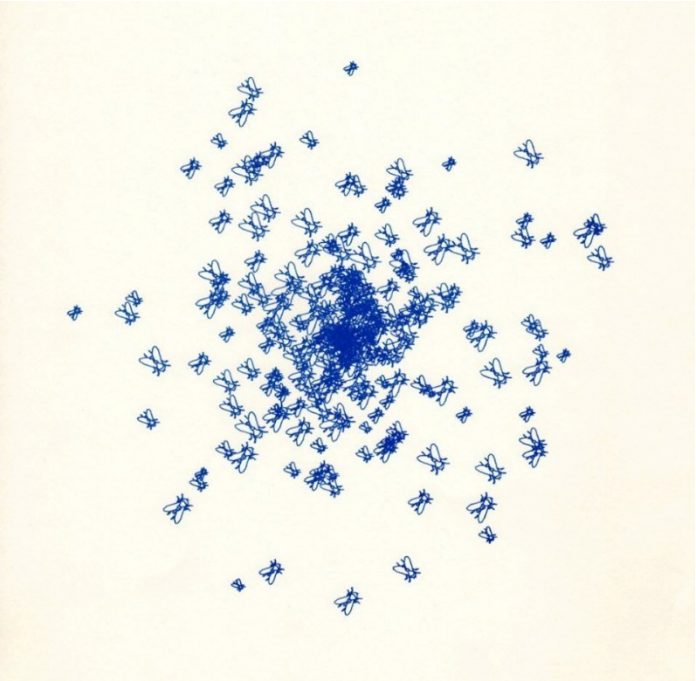
Každá muška v obraze má navíc své specifické vizuální vlastnosti – liší se velikostí a orientací. I tyto parametry byly stanoveny náhodně v rámci předem definovaných mezí. U natočení šlo zřejmě o náhodný výběr mezi dvěma úhly, zatímco velikost byla generována s mírnými odchylkami od základního měřítka, aby se zachoval jednotný vzhled, ale zároveň se zabránilo monotónnosti. Tak vznikla kompozice, ve které žádný objekt není zcela identický s jiným, ale všechny sdílejí stejný základní tvar. Tento princip opakování s variací je pro rané generativní umění typický.

Hotová data – tedy souřadnice, úhly natočení a měřítko každé mušky – byla následně zpracována výstupním zařízením, konkrétně mechanickým plotterem, který postupně kreslil jednotlivé obrazce na papír. Kresba nebyla vytvářena ručně, ale strojově, přičemž každý výskyt mušky byl transformován pomocí výpočtů do své konečné podoby na základě vstupních parametrů. Samotný tvar mušky pravděpodobně vznikl jako jednoduchý vektorový výkres, který se poté opakovaně vykresloval na různých pozicích a pod různými úhly.

Výsledkem je obraz, který i přes svůj algoritmický původ působí organicky a přirozeně. Rozmístění mušek vytváří dojem živého pohybu nebo přirozeného rozptylu částic, a přestože jsou jednotlivé prvky jednoduché, jejich kombinace vytváří komplexní strukturu. Právě v tomto spojení jednoduchého vizuálního jazyka s matematickou logikou spočívá umělecká síla díla. *Flies* je nejen estetickým artefaktem, ale i důkazem toho, že počítač – navzdory své rigidní povaze – může být nástrojem výtvarné kreativity.

Dílo tak ilustruje rané využití náhodnosti jako estetického principu a ukazuje, jak lze pomocí výpočetních metod generovat vizuální kompozice, které působí živěji než ručně kreslené struktury. V kontextu doby šlo o průkopnický akt, který otevíral možnosti nové formy tvorby, kde lidský umělecký záměr vstupuje do dialogu s výpočetním algoritmem.

„A computer program generates random numbers which determine the distribution of a specific number of flies in a series of 1" concentric rings. Within predetermined limits the random number generator also decides the orientation and the size of each fly.“ – Charles Csuri[29]

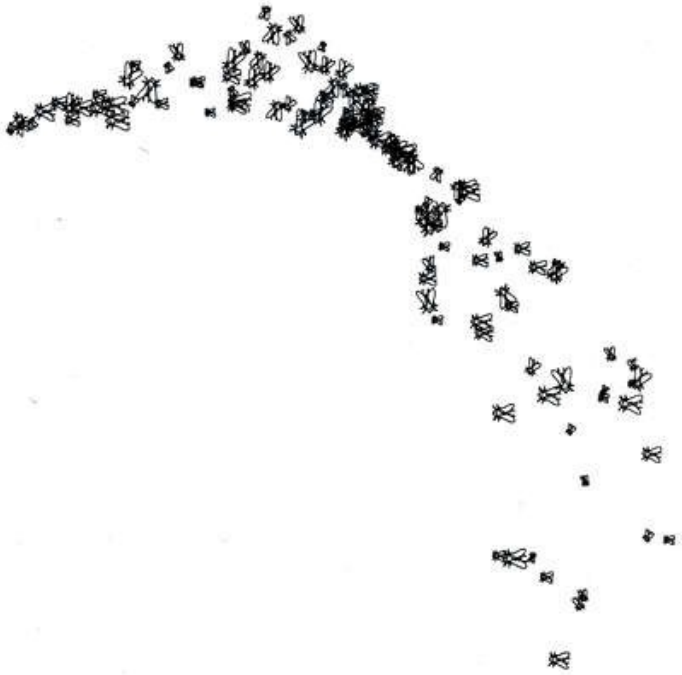


Obrázek 11 – Flies (1966) [17]

Analýza originálního díla Flies Transformed

Obraz *Flies Transformed* navazuje přímo na dílo *Flies*, ale zatímco první kompozice představovala vizuální výzkum náhodného rozmístění jednoduchých objektů v souřadném prostoru, *Flies Transformed* představuje další krok: aplikaci matematické transformace na již existující obrazovou strukturu. Zatímco ve *Flies* byla náhodnost určující pro samotné umístění a vlastnosti jednotlivých mušek, zde je zachována původní vstupní sada dat, avšak ta je podrobena cílené geometrické deformaci prostřednictvím matematických funkcí.

Tímto přístupem se autoři – zejména Charles Csuri – přiblížili myšlence, že počítač není pouze nástrojem generování dat, ale i prostředím pro transformaci reality do alternativních vizuálních forem. Samotný algoritmický základ zůstává podobný: původní souřadnice a vlastnosti objektů jsou známy, ale před vykreslením dochází k jejich transformaci podle matematických rovnic, které určují nové hodnoty souřadnic. Tyto rovnice mohou být lineární, ale v případě *Flies Transformed* šlo pravděpodobně o nelineární souřadnicovou transformaci, která výrazně mění prostorové vztahy mezi jednotlivými prvky.



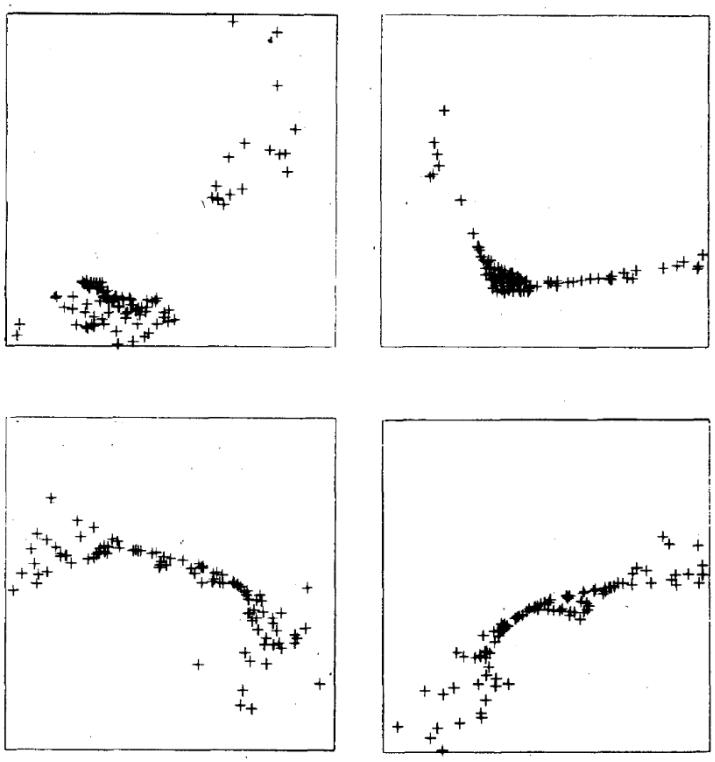
Obrázek 12 – Flies Transformed [18]

Podle článku *Art, Computers and Mathematics* (1968) autoři v tomto typu obrazů experimentovali s transformacemi ve tvaru:[29]

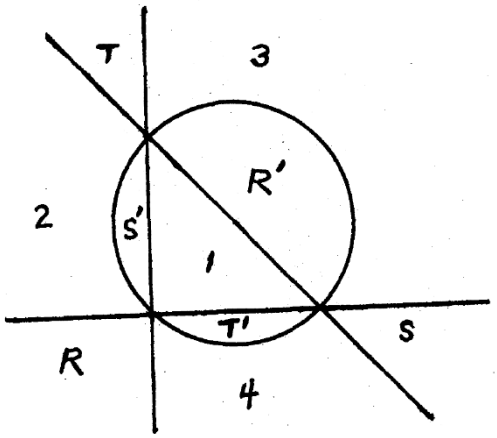
$$x' = \frac{y(x + y - 1)}{(x^2 + y^2 - x - y)},$$

$$y' = \frac{x(x + y - 1)}{(x^2 + y^2 - x - y)}.$$

„x′ = y(x + y — l)/(x² + y² - x - y) y′ = x* (x + y - l)/(x² + y² - x - y) cause the lettered primed and unprimed areas to interchange while the numbered areas map into themselves. To produce the accompanying figures rectangular areas were defined in which random coordinates were generated. After applying the transformation equations those coordinates which fell inside a second rectangle were accepted as the location for a fly or plus sign.“ – Charles Csuri[34]*



Obrázek 13 – Flies Transformed [34]



Obrázek 14 – Rozdělení sektorů [34]

Touto transformací se mění původní pozice každého bodu ve vztahu k jeho vlastním souřadnicím, čímž dochází k nepravidelnému zakřivení a protažení původní struktury. Výsledkem je obraz, ve kterém původní kruhové uspořádání mušek mizí a je nahrazeno složitější, těžko předvídatelnou sítí zakřivených trajektorií a zhuštěných oblastí. Některé prvky jsou navzájem „přesunuty“ mezi oblastmi, což souvisí s myšlenkou topologické výměny obrazových segmentů.

Z hlediska postupu byla transformace aplikována až po vygenerování původní sady dat. To znamená, že náhodně určené souřadnice, rotace a měřítka mušek zůstaly stejné, ale místo aby byly vykresleny na své původní pozici, byly převedeny do nové pozice výpočtem nových (x',

y') souřadnic podle výše uvedené rovnice. Ostatní parametry jako orientace a velikost zřejmě zůstaly beze změny, což umožňuje divákovi vnímat identitu jednotlivých mušek i v deformovaném poli.

Tímto způsobem se obraz stává výsledkem intervence matematického systému do organické struktury. Z původně vyvážené a rozprostřené kompozice vzniká dynamická deformace, která působí téměř jako fyzikální jev – jako by byly mušky přetahovány gravitačními silami nebo nasávány do víru. Právě v této schopnosti transformace narušit stabilitu, a přesto zachovat čitelnost jednotlivých prvků spočívá síla kompozice.

Výsledný obraz tak představuje nejen nový vizuální zážitek, ale také konceptuální posun: místo generování náhodných dat je hlavní kreativní složkou matematická manipulace s existujícím obrazovým prostorem. Tento přístup otevírá cestu k využití formálních systémů nejen jako prostředků kompozice, ale i jako nástroje transformace reality do alternativních vizuálních struktur.

Z hlediska počítačového zpracování znamenala tato změna i vyšší nároky na výpočet – bylo třeba aplikovat transformaci na původní souřadnice a poté znovu vypočítat výslednou pozici. V kontextu tehdejší výpočetní techniky (děrné štítky, dávkové zpracování, plottery) jde o značně komplexní operaci, která svědčí o hlubokém porozumění autorů jak matematice, tak vizuální kompozici.

Struktura a inicializace scény

Po spuštění části aplikace *Flies* probíhá automatická inicializace všech grafických a logických komponent, která zajišťuje, že je scéna připravena k okamžitému použití bez nutnosti ručního zásahu uživatele. Tato inicializace probíhá v metodě *initialize(),* která je volána JavaFX systémem ve chvíli, kdy se načte odpovídající FXML soubor.

V rámci tohoto procesu jsou nejprve nastaveny výchozí hodnoty ovládacích prvků. Volba osy transformace je předvyplněna možnostmi „X“ a „Y“, přičemž výchozí hodnota je nastavena na „X“. Posuvník určující hraniční hodnotu pro výběrovou transformaci je inicializován na střední pozici, tedy 0.5. Tyto hodnoty zajišťují, že při prvním spuštění scény bude zobrazen výstup, který má smysluplnou a stabilní strukturu.

Součástí inicializace je také propojení jednotlivých grafických komponent s logikou aplikace prostřednictvím tzv. posluchačů změn. Tyto posluchače umožňují aplikaci reagovat na jakékoli úpravy velikosti panelů, změny hodnot posuvníků nebo stavů přepínačů. Například při změně velikosti výkresového panelu je okamžitě vyvolána metoda *createPaintings(),* která provede nové vykreslení simulace s ohledem na aktuální rozměry. Podobně i změna hodnoty posuvníku nebo přepnutí režimu transformace automaticky spustí překreslení bez nutnosti potvrzení uživatelem. Díky tomu je celá aplikace velmi dynamická a interaktivní.

Významnou součástí inicializační fáze je také provedení první vizualizace. Aplikace ihned po načtení komponent a nastavení výchozích parametrů spustí metodu *createPaintings(),* která podle předvolených hodnot provede vygenerování určitého počtu mušek a jejich případnou transformaci. Uživatel tak po otevření okna okamžitě vidí výstup a může s ním dále

experimentovat. Tento přístup výrazně zvyšuje použitelnost programu, protože eliminuje „prázdný stav“ a poskytuje zpětnou vazbu okamžitě po spuštění.

Metoda inicializace vypadá takto:

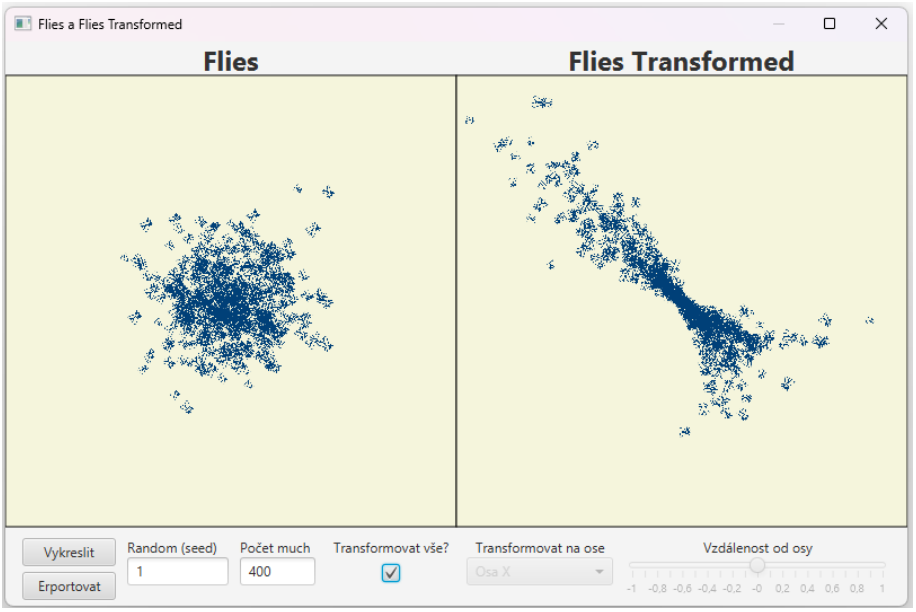
```
public void initialize(URL url, ResourceBundle resourceBundle) {
    // Reakce na změnu velikosti panelu – překreslit mouchy
    paneFlies.widthProperty().addListener(cl -> createPaintings());
    paneFlies.heightProperty().addListener(cl -> createPaintings());

    // Reakce na změnu checkboxu "Transformovat všechny"
    cbTransformAll.selectedProperty().addListener((obs, oldV, newV) -> {
        if (newV) {
            cbAxis.setDisable(true);
            slOffset.setDisable(true);
        } else {
            cbAxis.setDisable(false);
            slOffset.setDisable(false);
        }
        createPaintings();
    });

    // Naplnění volby os pro transformaci a nastavení výchozí hodnoty
    cbAxis.getItems().add("Osa X");
    cbAxis.getItems().add("Osa Y");
    cbAxis.getSelectionModel().select(0);

    // Reakce na změnu zvolené osy
    cbAxis.getSelectionModel().selectedItemProperty().addListener((obs,
oldV, newV) -> {
        if (newV != null) {
            createPaintings();
        }
    });

    slOffset.valueProperty().addListener((obs, oldVal, newVal) ->
createPaintings());

    // Výchozí hodnoty
    seed = 1;
    count = 400;

    // Prvotní vykreslení
    createPaintings();
}
```

Tato automatická inicializace vytváří základ pro celý interaktivní proces práce se simulací. Uživatel může ihned upravovat vstupy, měnit rozměry výkresu, aplikovat transformace a sledovat dopady svých rozhodnutí v reálném čase. Z technického hlediska tak inicializace nejen připravuje scénu, ale nastavuje i vazby, které umožňují celou její dynamiku.

Uživatelské vstupy a jejich validace

Uživatelské vstupy v části aplikace *Flies* slouží k parametrizaci generovaného obrazu a umožňují ovlivnit klíčové aspekty simulace. Uživatel má možnost zadat několik vstupních hodnot prostřednictvím prvků grafického rozhraní. Dvě hlavní textová pole slouží pro zadání celého čísla určujícího počet generovaných mušek a pro zadání tzv. *seed* hodnoty, která

inicializuje generátor pseudonáhodných čísel. Díky použití vlastního seedu lze zajistit opakovatelnost výstupu, což je důležité jak z hlediska ladění, tak pro konzistentní porovnávání transformací při stejných datech. Dalším prvkem je zaškrtávací políčko, které umožňuje přepnout mezi dvěma režimy: buď budou transformovány všechny mušky, nebo pouze ty, které spadají do vybrané části obrazového prostoru. V případě selektivní transformace může uživatel nastavit osu (X nebo Y), vůči níž bude transformace rozhodována, a také hodnotu posunu, která určuje mezní hranici výběru.



Obrázek 15 – Okno vlastní aplikace Flies a Flies Transformed

Zadání těchto hodnot je podrobováno validaci, aby bylo zajištěno, že program bude pracovat s korektními a smysluplnými daty. Vstupy z textových polí jsou převáděny na číselné hodnoty pomocí standardních metod, přičemž se zachycuje případná výjimka *NumberFormatException*. Pokud dojde k zadání nečíselné hodnoty, prázdného řetězce nebo jiné nevalidní hodnoty, aplikace nezkolabuje, ale zobrazí chybové dialogové okno s informací o tom, že vstupní hodnota není platné číslo. Tímto způsobem je uživatel srozumitelně informován o problému a je mu umožněno vstup opravit. Program tím zajišťuje robustnost a uživatelskou přívětivost i v případě chybných nebo neúplných vstupů. Tento princip validace přispívá ke stabilitě celé aplikace, protože minimalizuje riziko výpadků nebo nepředvídatelného chování v důsledku chybných dat. Validace vstupů tak hraje důležitou roli nejen z pohledu technického zabezpečení, ale i jako součást dobrého návrhu interakce mezi uživatelem a aplikací.

Metoda pro validaci uživatelských vstupů vypadá takto:

```
public void btnDrawOnAction() {
    try{
        seed = Long.parseLong(tfRandom.getText());
        count = Integer.parseInt(tfCount.getText());

        //
        if(count < 1){
```

```
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Chybný vstup");
                alert.setHeaderText("Nulový nebo záporný počet mouch");
                alert.setContentText("Zadej počet much který bude 1 nebo větší
");
                alert.showAndWait();
            }

            // Nové vykreslení
            createPaintings();
        }catch (NumberFormatException ex) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Chybný vstup");
            alert.setHeaderText("Neplatné číslo");
            alert.setContentText("Zadejte platné číslo. " + ex.getMessage());
            alert.showAndWait();
        }
    }
```

Generování pseudonáhodných prvků

Generování náhodných dat v části aplikace *Flies* tvoří základní mechaniku, která určuje prostorové rozložení a vzhled jednotlivých mušek ve vizualizaci. Všechny náhodné hodnoty jsou generovány pomocí standardního pseudonáhodného generátoru čísel, který je inicializován hodnotou zadanou uživatelem prostřednictvím vstupního pole. Tato tzv. *seed* hodnota zajišťuje deterministické chování generátoru – pokud uživatel zadá stejný seed, výstup bude vždy stejný, což umožňuje opakovatelnost experimentu a srovnání mezi jednotlivými scénami nebo transformacemi.

Každá muška je generována samostatně v rámci cyklu, který probíhá podle zadaného počtu. Pro určení její pozice jsou použita dvě náhodná čísla reprezentující souřadnice v normovaném dvourozměrném prostoru. Tato čísla nejsou vybírána rovnoměrně, ale podle Gaussova (normálního) rozdělení. Použitím funkce *nextGaussian()* se získávají hodnoty, které přirozeně koncentrují body blíže ke středu prostoru, čímž vzniká efekt podobný přirozenému shluku – právě takový, jaký je charakteristický pro původní dílo *Flies*. Aby byly hodnoty převedeny do rozsahu mezi 0 a 1, je výstup z generátoru posunut a škálován. Výsledné hodnoty pak představují relativní souřadnice v rámci výkresové plochy a následně jsou přepočítány na absolutní pozice podle aktuální velikosti výstupního panelu.

Kromě pozice každé mušky je náhodně generována i její velikost a rotace. Velikost je určována jako procentuální škálování původního tvaru a pohybuje se v definovaném rozsahu, například mezi 50 % a 100 %. Rotace je vybírána náhodně ze dvou nebo více předdefinovaných úhlů, což zajišťuje vizuální variabilitu a zabraňuje opakování identického vzhledu u všech prvků. Tyto variace přispívají k organickému dojmu výsledného obrazu, a přestože každý tvar vychází ze stejného obrázku mušky, díky rozdílům v poloze, velikosti a orientaci působí každý výskyt unikátně. Celkově tedy generování náhodných dat v této části programu kombinuje deterministickou strukturu (danou seedem) s řízenou variabilitou, která napodobuje přirozené rozptylové jevy a odpovídá estetickému záměru původního generativního díla.

Kód pro vygenerování náhodné pozice, náhodné velikost a následné vytvoření mouchy a výběru náhodné rotace ze dvou předdefinovaných vypadá takto:

```
// Generování náhodné pozice s normálním rozdělením
double randomX = ((random.nextGaussian() + 3) / 6);
double randomY = ((random.nextGaussian() + 3) / 6);

double x = midX + ((randomX - 0.5) * (midX));
double y = midY + ((randomY - 0.5) * (midX));

// Náhodná velikost
double size = 0.5 + (random.nextDouble() * 0.5);
double flyWidth = (67.7 / (1200 / midX)) * size;
double flyHeight = (83.5 / (1200 / midX)) * size;

// Vytvoření původní mouchy
Rectangle fly1 = new Rectangle(x - (flyWidth / 2), y - (flyHeight/2),
flyWidth, flyHeight);    // původní
fly1.setFill(imagePattern);
fly1.setRotate(random.nextBoolean() ? 45 : 135);
paneFlies.getChildren().add(fly1);
```

Matematická transformace souřadnic

Matematická transformace souřadnic v části aplikace *Flies* slouží k vytvoření alternativního pohledu na původně generovaná data a je inspirována nelineárními transformačními funkcemi, které byly použity v původních experimentech Charlese Csuriho. Transformace se aplikuje na souřadnice jednotlivých mušek a jejím cílem je vytvořit novou vizuální strukturu, která vychází z existujícího rozložení, ale deformuje jej podle přesně definovaných matematických pravidel. V praxi to znamená, že každá původní dvojice souřadnic, která vznikla náhodným generováním, je převedena do nového bodu podle transformační rovnice. Tento převod je proveden bez změny měřítka nebo orientace mušky, ale výhradně změnou její polohy v prostoru.

Použitá transformace pracuje s hodnotami v normalizovaném intervalu 0 až 1 a je definována vzorcem, ve kterém jsou nové souřadnice počítány na základě původních hodnot a jejich vzájemné kombinace. Konkrétně se používá vzorec, který počítá s výrazem uvedeným v analýze díla *Flies Transformed*, a výsledkem jsou nové hodnoty (x′, y′), jež představují zakřivenou deformaci původního rozložení. Tato rovnice má nelineární charakter, což znamená, že neprovádí jednoduché posunutí nebo zvětšení, ale složitější zakřivení souřadnicového prostoru. Výsledkem je efekt, kdy jsou některé oblasti zahuštěny, jiné roztaženy a celková kompozice působí opticky deformovaně jakoby roztažená v prostoru nebo zakřivená kolem neviditelné osy.

V aplikaci se tato transformace provádí nad každou muškou zvlášť, a to až po jejím vygenerování. Původní náhodně určené hodnoty souřadnic jsou dosazeny do transformační rovnice, a výstupem jsou nové relativní hodnoty, které jsou následně přepočteny na absolutní pozice v pravém výkresovém panelu. Transformace se přitom nemusí vztahovat na všechny mušky – uživatel si může zvolit, zda má být transformována celá množina, nebo pouze část, která splňuje určitou podmínku (například leží na pravé straně podle zvolené osy a hodnoty

posuvníku). Tato volba ovlivňuje výběr dat, která budou transformována, ale nikoli samotnou matematickou funkci.

Kód pro transformaci souřadnic vypadá takto:

```
// Výpočet transformované pozice pomocí matematického vzorce
double formula = (randomX + randomY - 1) / (Math.pow(randomX, 2) +
Math.pow(randomY, 2) - randomX - randomY);
double x1 = randomY * formula;
double y1 = randomX * formula;

double transformedX = (x1 * (midX));
double transformedY = (y1 * (midX));
```

Díky této transformaci vzniká vizuální kontrast mezi levým panelem, který zobrazuje původní rozložení, a pravým panelem, kde jsou prvky deformovány podle matematického modelu. Tento rozdíl umožňuje nejen estetické srovnání, ale také přímou demonstraci vlivu matematických operací na prostorové vztahy mezi objekty. Celý proces probíhá v reálném čase a je plně automatizovaný, přičemž zachovává vazbu mezi původní a transformovanou verzí každé jednotlivé mušky.

Výběr transformace

Výběr transformace v části *Flies* rozšiřuje základní transformační mechanismus o možnost selektivně určovat, které mušky budou transformovány a které zůstanou nezměněné. Tato funkcionalita přináší uživateli možnost ovlivnit, zda bude transformace aplikována na všechny generované prvky, nebo pouze na jejich podmnožinu vybranou podle prostorových kritérií. Rozhodujícími parametry pro tento výběr jsou osa (X nebo Y) a hodnota posuvníku, který určuje prahovou mez. Uživatel si nejprve zvolí, zda má být aktivována možnost selektivní transformace. Pokud je tato volba potvrzena, aplikuje se transformace pouze na ty mušky, jejichž normalizovaná souřadnice odpovídající zvolené ose je větší nebo rovna hodnotě nastavené posuvníkem.

Celý proces probíhá v reálném čase při vykreslování scén. Pro každou mušku se zkontroluje, zda splňuje podmínku výběru: pokud ano, je na ni aplikována transformace a je vykreslena na pravém panelu; pokud ne, zůstává nezměněná a na pravé straně se nezobrazí. Uživatel tak může například zvolit transformaci pouze těch mušek, které se nacházejí ve spodní polovině obrazu (při výběru osy Y a hodnotě posuvníku blízké 0.5), nebo naopak aplikovat transformaci pouze na krajní pravý sloupec (při výběru osy X a hodnotě 0.9). Vizuálně je výběrová oblast zobrazena pomocí zvýrazněných čar a poloprůhledného obdélníku v původním panelu, což poskytuje přímou zpětnou vazbu o tom, která část dat bude podrobena transformaci.

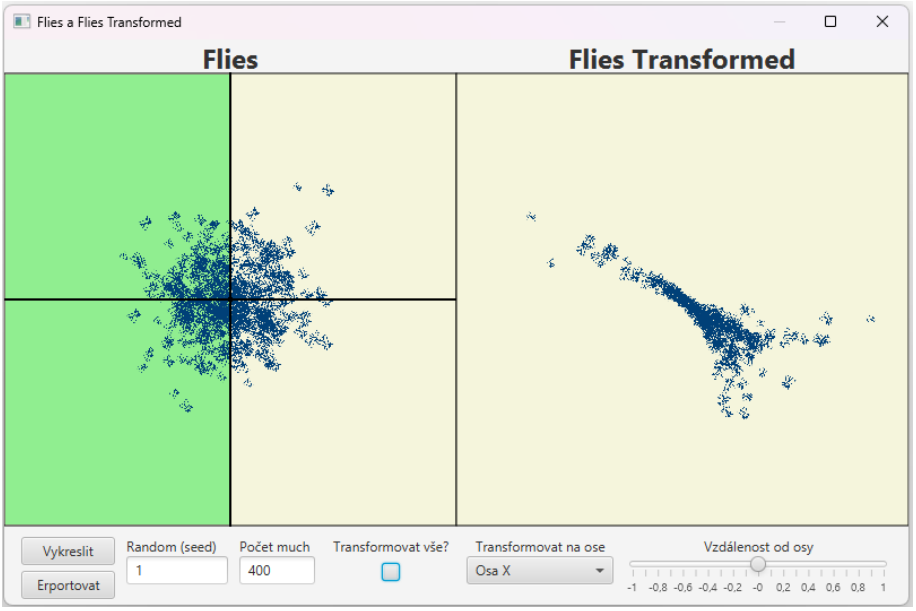
Kód pro selektivní výběr transformace vypadá takto:

```
// Kontrola, zda se transformovaná moucha nachází v zobrazitelné oblasti
if (((transformedX > -(midX - (x0 + 10))) && (transformedX < (midX - (x0 +
10)))) &&
        ((transformedY > -(midY - (y0 + 10))) && (transformedY < (midY -
(y0 + 10))))){
```

```
    if(cbTransformAll.isSelected()){
        // Transformuj všechny mouchy bez ohledu na osu
        paneFliesTransformed.getChildren().add(fly2);
    } else {
        // Transformuj podle vybrané osy
        switch (cbAxis.getSelectionModel().getSelectedIndex()){
            // Osa X
            case 0:
                line1 = new Line((midX + ((editedOffset) * (midX))), y0,
(midX + ((editedOffset) * (midX))), (2 * midY) - y0);
                line2 = new Line(x0, (midY + ((-editedOffset) * (midY))),
(2 * midX) - x0, (midY + ((-editedOffset) * (midY))));
                line3 = new Line((midX + ((editedOffset) * (midX))), (midY
+ ((editedOffset) * (midY))), (midX + ((-editedOffset) * (midX))), (midY +
((-editedOffset) * (midY))));

                // Filtrace podle offsetu
                if ((randomX < offset) && (offset <= 0.5)){
                    paneFliesTransformed.getChildren().add(fly2);
                    // Filtrace podle offsetu
                    representation = new Rectangle(x0+1, y0+1, (midX +
((editedOffset) * (midX))) - x0, (midY - y0 - 1)*2);
                } else if ((randomX > offset) && (offset > 0.5)){
                    paneFliesTransformed.getChildren().add(fly2);
                    // Filtrace podle offsetu
                    representation = new Rectangle((midX + ((editedOffset)
* (midX)))+1, y0+1, (midX - ((editedOffset) * (midX))) - x0 - 1, (midY - y0
- 1)*2);
                }
                break;
            // Osa Y
            case 1:
                line1 = new Line(x0, (midY + ((editedOffset) * (midY))), (2
* midX) - x0, (midY + ((editedOffset) * (midY))));
                line2 = new Line((midX + ((editedOffset) * (midX))), y0,
(midX + ((editedOffset) * (midX))), (2 * midY) - y0);
                line3 = new Line((midX + ((editedOffset) * (midX))), (midY
+ ((-editedOffset) * (midY))), (midX + ((-editedOffset) * (midX))), (midY +
((editedOffset) * (midY))));

                // Filtrace podle offsetu
                if ((randomY < offset) && (offset <= 0.5)){
                    paneFliesTransformed.getChildren().add(fly2);
                    // Filtrace podle offsetu
                    representation = new Rectangle(x0 + 1, y0+1, (midX - x0
- 1)*2, (midY + ((editedOffset) * (midY))) - y0);
                } else if ((randomY > offset) && (offset > 0.5)){
                    paneFliesTransformed.getChildren().add(fly2);
                    // Filtrace podle offsetu
                    representation = new Rectangle(x0 + 1, (midY +
((editedOffset) * (midY)))+1, (midX - x0 - 1)*2, (midY - ((editedOffset) *
(midY))) - y0 - 2);
                }
                break;
        }
        // Vykreslení pomocných linií a oblasti
        line1.setStrokeWidth(2);
        paneFlies.getChildren().add(line1);
        line2.setStrokeWidth(2);
        paneFlies.getChildren().add(line2);
        line3.setStrokeWidth(3);
```

```
            paneFlies.getChildren().add(line3);

            representation.setFill(Color.LIGHTGREEN);
            paneFlies.getChildren().add(1, representation);
        }
}
```

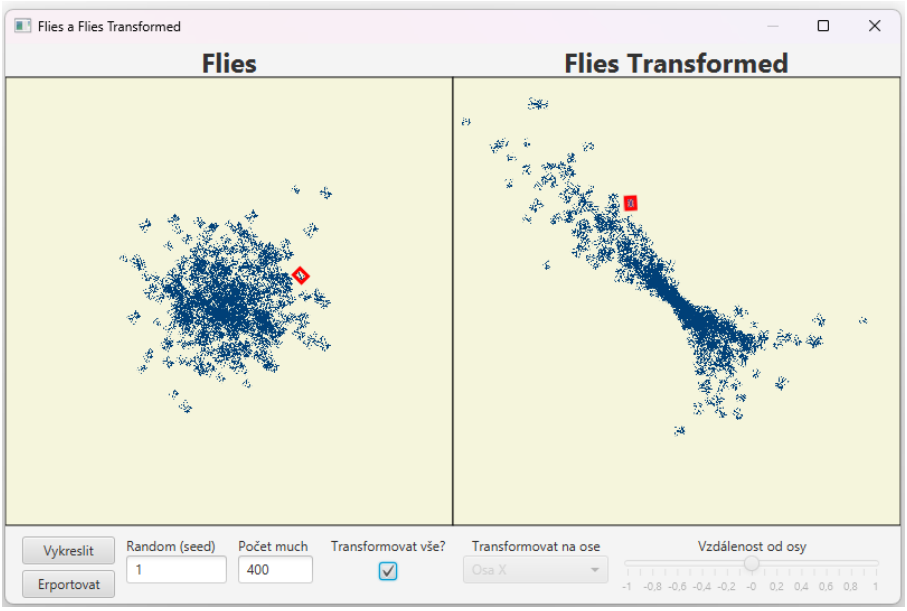
Tento mechanismus posouvá aplikaci z čistě generativního nástroje směrem k interaktivnímu experimentu, v němž může uživatel zkoumat účinek transformací na různé části dat. Zároveň umožňuje vytvářet výrazně odlišné kompozice i při stejném výchozím rozložení much, což rozšiřuje tvůrčí možnosti programu. Logika výběrové transformace je plně integrována do procesu vykreslování a nijak nezatěžuje uživatele složitými operacemi – stačí jednoduché nastavení parametrů v rozhraní a aplikace automaticky vykreslí odpovídající obraz.



Obrázek 16 – Aplikace při výběru osy X a hodnotě posuvníku blízké 0

Interaktivita ve vizualizaci

Interaktivita ve vizualizaci mušek v části aplikace *Flies* je rozšířena o funkci zvýraznění souvisejících prvků, která umožňuje uživateli lépe porozumět vztahu mezi původní a transformovanou podobou každé jednotlivé mušky. Každý pár mušek – jedna ve výchozí podobě na levé straně a druhá v transformované podobě na pravé straně – tvoří logicky propojenou dvojici. Aby bylo možné tyto dvojice snadno identifikovat, je implementována funkce zvýraznění, která reaguje na pohyb kurzoru myši. Při přejetí nad kteroukoli muškou v původním obraze dojde k vizuálnímu zvýraznění jak této mušky, tak jejího protějšku ve scéně transformované. Zvýraznění je realizováno aplikací optického efektu, který zvýší jas nebo přidá záři, a současně barevného orámování objektu. Stejná logika platí i v opačném směru – pokud uživatel přejede myší nad transformovanou muškou, zvýrazní se také její originál. Tímto způsobem lze velmi rychle a intuitivně sledovat, jak byla pozice konkrétního prvku ovlivněna transformací.



Obrázek 17 – Aplikace Flies při zvýraznění much

Celý mechanismus je implementován pomocí obslužných metod reagujících na události *onMouseEntered* a *onMouseExited*, které se registrují ke každé instanci mušky při jejím vytvoření. Tyto události nastavují nebo ruší vizuální efekty v reálném čase, čímž je dosaženo plynulého interaktivního dojmu. Z uživatelského hlediska tak vzniká velmi srozumitelný způsob, jak mezi sebou porovnat odpovídající dvojice prvků. Tato interakce podporuje nejen estetické vnímání proměny obrazu, ale i analytické zkoumání změn v souřadnicovém systému, což je přínosné například při výuce transformací nebo při demonstraci účinků matematických funkcí na datovou množinu. Výsledkem je nástroj, který spojuje výtvarný a matematický pohled a umožňuje hlubší porozumění vztahu mezi původní strukturou dat a její transformovanou podobou.

Kód pro zvýraznění much vypadá takto:

```
// Efekt po přejetí myší – zvýrazní spárované mouchy
private void addHighlightingEffect(Rectangle fly1, Rectangle fly2) {
    fly1.setOnMouseEntered(e -> {
        fly1.setEffect(new Glow(0.8));
        fly1.setStroke(Color.RED);
        fly1.setStrokeWidth(3);
        fly2.setEffect(new Glow(0.8));
        fly2.setStroke(Color.RED);
        fly2.setStrokeWidth(3);
    });
    fly1.setOnMouseExited(e -> {
        fly1.setEffect(null);
        fly1.setStroke(null);
        fly1.setStrokeWidth(0);
        fly2.setEffect(null);
        fly2.setStroke(null);
        fly2.setStrokeWidth(0);
    });
}
```

Exportování obrazů ve vysoké kvalitě

Exportování obrázků v aplikaci *Flies* je promyšleně navrženou součástí, která uživateli umožňuje uložit výsledek vizualizace ve vysoké kvalitě. Tato funkce je navržena tak, aby bylo možné exportovat nejen běžný pohled na obrazovku, ale i obraz s podstatně vyšším rozlišením, který je vhodný například pro tisk nebo prezentaci.

Export je dostupný skrze tlačítko „Exportovat“, po jehož stisknutí se otevře dialog, kde si uživatel zvolí, zda chce uložit *původní mouchy*, *transformované mouchy* nebo *obojí*. V případě volby *obojí* aplikace vygeneruje dva samostatné obrázky – každý pro jinou část vizualizace – a přidá k jejich názvům příslušnou příponu (_puvodni, _transformovane), aby nedošlo k jejich přepsání.

Při samotném exportu dochází k vytvoření tzv. snapshotu panelu, který je roztažen pomocí transformační matice na výrazně větší rozměr – výchozí škálovací faktor je 10×, ale program dynamicky zvyšuje měřítko tak, aby výsledná šířka obrázku přesáhla 10 000 pixelů. To zajišťuje, že i na velkoformátovém výtisku zůstane obraz detailní a ostrý. Výsledný snímek je pak uložen ve formátu PNG, a to včetně průhledného pozadí, což umožňuje další zpracování v externích grafických programech.

Export je doprovázen kontrolními mechanismy – pokud dojde k chybě (například při ukládání na nedostupné místo), je uživatel informován prostřednictvím chybového hlášení. V opačném případě se zobrazí potvrzení o úspěšném dokončení operace. Funkcionalita je přístupná z jakéhokoli stavu aplikace a nevyžaduje žádné další zadání – vždy vychází z aktuálně vykreslené vizualizace.

Tento systém exportu doplňuje interaktivní charakter aplikace o důležitý výstupní kanál a umožňuje nejen sdílení generovaných děl, ale i jejich archivaci a tisk. Díky možnosti volby rozsahu exportu a vysokému rozlišení je výstup plnohodnotným reprezentantem generativního umění v digitální i fyzické podobě.

Kód pro exportování obrazů vypadá takto:

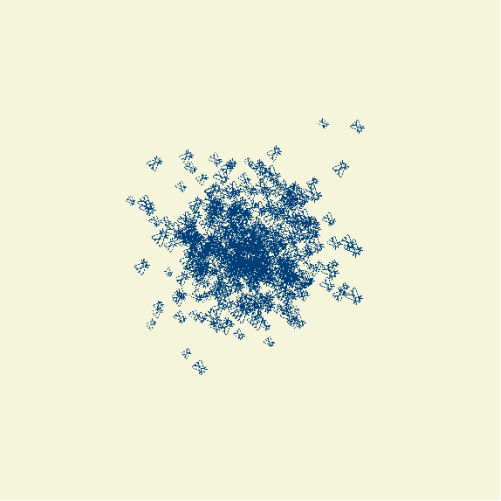
```
// Akce po stisknutí tlačítka "Exportovat"
public void btnExportOnAction() {
    // Výběr panelu pro export
    List<String> options = Arrays.asList("Původní mouchy", "Transformované
mouchy", "Obojí");
    ChoiceDialog<String> dialog = new ChoiceDialog<>("Původní mouchy",
options);
    dialog.setTitle("Výběr exportu");
    dialog.setHeaderText("Vyber, co chceš exportovat:");
    dialog.setContentText("Panel:");

    Optional<String> result = dialog.showAndWait();
    if (result.isEmpty()) return;

    // Výběr cílového souboru
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Uložit obrázek");
    fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("PNG obrázky", "*.png"));
    fileChooser.setInitialFileName("Flies.png");
```

```
    File file =
fileChooser.showSaveDialog(btnExport.getScene().getWindow());
    if (file == null) return;

    double scale = 10.0;

    try {
        // Export původních much
        if (result.get().equals("Původní mouchy") ||
result.get().equals("Obojí")) {
            exportPaneToPNG(paneFlies, file, scale,
result.get().equals("Obojí") ? "_puvodni" : "");
        }

        // Export transformovaných much
        if (result.get().equals("Transformované mouchy") ||
result.get().equals("Obojí")) {
            exportPaneToPNG(paneFliesTransformed, file, scale,
result.get().equals("Obojí") ? "_transformovane" : "");
        }

        // Zobrazení úspěšného hlášení
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Export dokončen");
        alert.setHeaderText(null);
        alert.setContentText("Export úspěšně dokončen.");
        alert.showAndWait();

    } catch (IOException e) {
        // Zobrazení chyby při exportu
        e.printStackTrace();
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Chyba při exportu");
        alert.setHeaderText("Nepodařilo se uložit obrázek");
        alert.setContentText(e.getMessage());
        alert.showAndWait();
    }
}

// Exportuje obsah zadaného JavaFX panelu do PNG obrázku
private void exportPaneToPNG(Pane pane, File file, double scale, String
suffix) throws IOException {
    double width = pane.getWidth();
    double height = pane.getHeight();

    // Pokud je export stále menší než 10000 px na šířku, zvětšujeme
měřítko
    while (width * scale < 10000){
        scale++;
    }

    // Nastavení parametrů pro snapshot – aplikujeme škálování a průhledné
pozadí
    SnapshotParameters params = new SnapshotParameters();
    params.setTransform(Transform.scale(scale, scale));
    params.setFill(Color.TRANSPARENT);

    // Vytvoření a naplnění obrázku snapshotem panelu
    WritableImage image = new WritableImage((int)(width * scale),
(int)(height * scale));
```

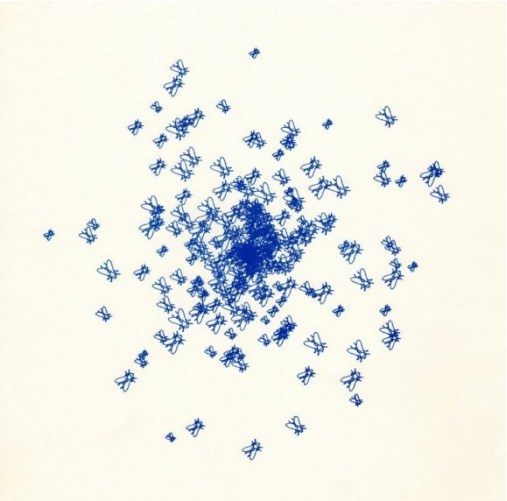
```
    pane.snapshot(params, image);

    // Úprava názvu souboru podle přípony
    String filePath = file.getAbsolutePath();
    if (suffix != null && !suffix.isEmpty()) {
        int dotIndex = filePath.lastIndexOf(".");
        if (dotIndex != -1) {
            // Vložíme příponu mezi název a koncovku (např.
export_transformovane.png)
            filePath = filePath.substring(0, dotIndex) + suffix +
filePath.substring(dotIndex);
        } else {
            filePath += suffix;
        }
    }

    // Uložení obrázku na disk jako PNG
    ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png", new
File(filePath));
}
```

Výstup a srovnání s originálem

Výstup aplikace ve své základní podobě (*Flies*) velmi věrně napodobuje vizuální charakter původního díla Charlese Csuriho z roku 1966. V obrázku č. 18 je patrná kompozice sestávající ze stovek mušek, které jsou rozmístěny s vyšší hustotou ve středu a s postupným rozptylem směrem k okrajům, přičemž každá z mušek má mírně odlišnou velikost a orientaci. Tento efekt vzniká použitím Gaussova rozdělení při generování pozic, stejně jako u originálu (obrázek č. 19). Výsledkem je obraz, který si zachovává statistický charakter a organickou estetiku předlohy. V porovnání s originálem je však zřejmé několik odlišností: aplikace pracuje s vyšším počtem mušek, což vede k větší hustotě ve středu a vizuálně sytějšímu výsledku. Zároveň jsou v simulaci mušky vykreslovány grafickými objekty s jemnou texturou, zatímco originál využíval kresby lineárního stylu, přičemž jednotlivé objekty se výrazněji liší tvarem. Kompozičně se ale výstup velmi přibližuje záměru autora – vzniká středově souměrné rozložení bez formální mřížky, kde dominují náhodně orientované, ale prostorově sjednocené prvky.

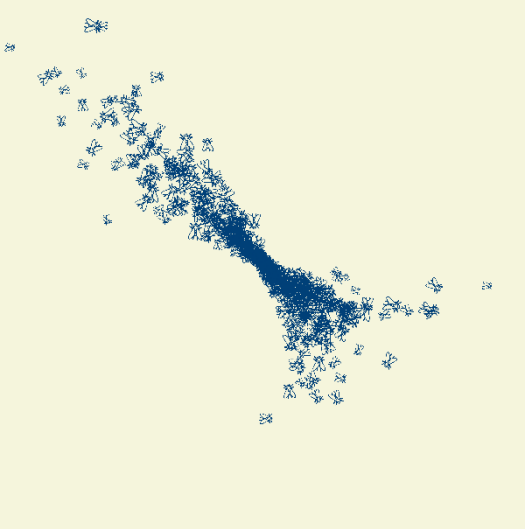


Obrázek 18 – Flies (vytvořen vlastní aplikací)

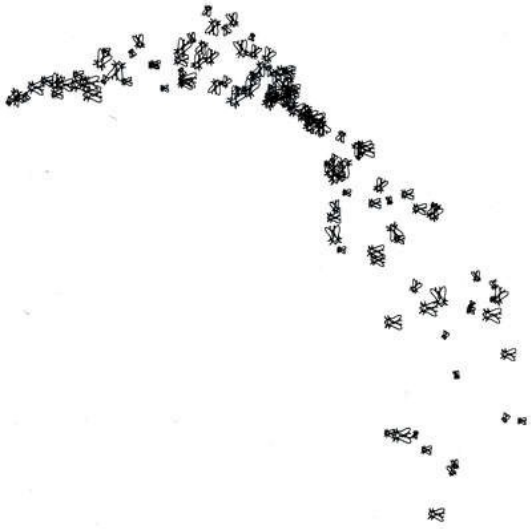


Obrázek 19 – Flies (1966) [17]

Ve obrázku č 20 je zachycen výstup transformované verze (*Flies Transformed*), kde jsou původní souřadnice každé mušky převedeny pomocí matematické rovnice do nového souřadnicového systému. Výsledkem je dynamická kompozice, v níž došlo k viditelné deformaci a zakřivení obrazce. Objekty se zhušťují do úzkého pásu uprostřed a rozšiřují se na obou koncích do jemného rozptylu. Tato vizuální deformace odpovídá účinku nelineární transformace, která mění vztahy mezi souřadnicemi a narušuje původní kruhové rozložení. Křivka, kterou mušky sledují, se velmi podobá trajektorii zobrazené v obrázku č. 21, jenž zachycuje původní *Flies Transformed* od Csuriho. I zde je patrné zakřivení a postupný rozptyl prvků, tentokrát však v řidším a ručněji působícím provedení. Původní obraz je méně nasycený a mušky jsou kresleny hrubším způsobem, což odpovídá technickým možnostem tehdejších plotterů.



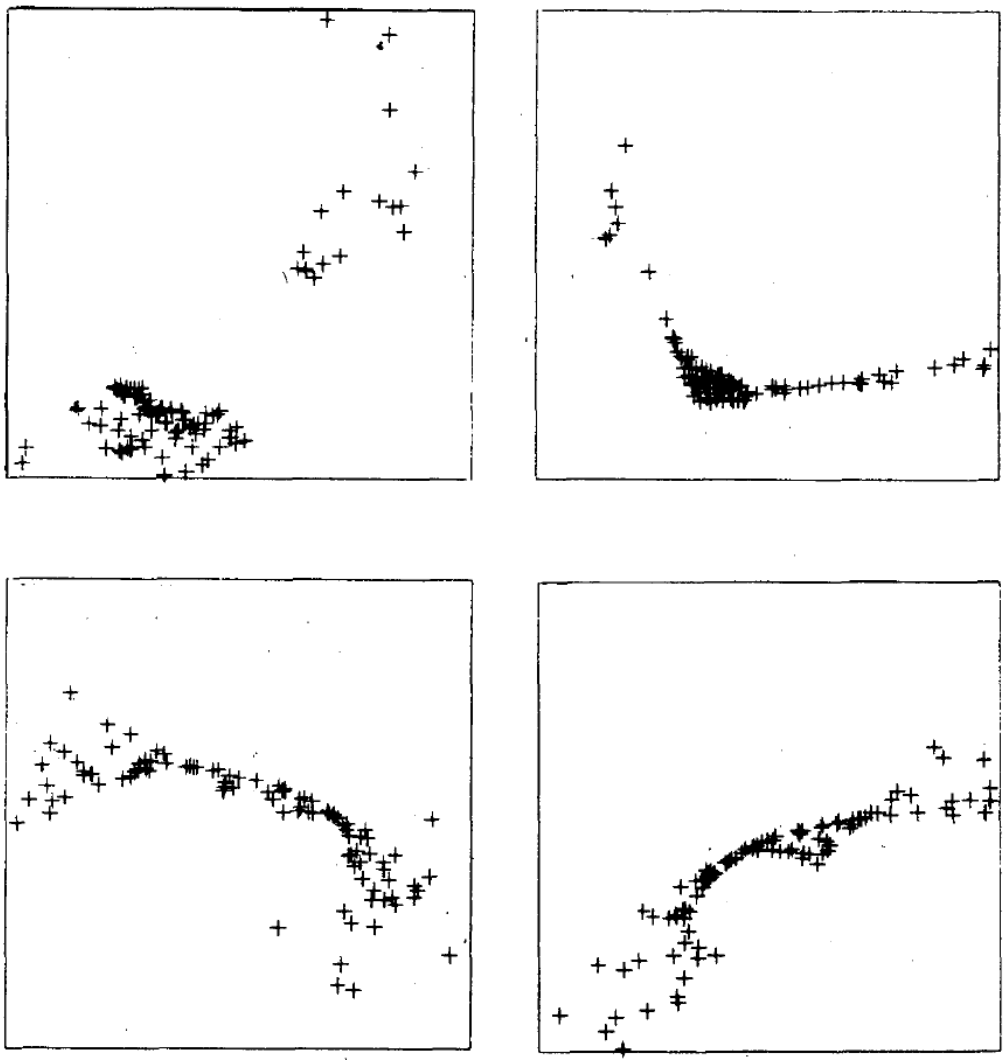
Obrázek 20 – Flies Transformed (vytvořen vlastní aplikací)



Obrázek 21 – Flies Transformed [18]

Další porovnání je porovnání obrazů s originálem z článku *Art, Computers and Mathematics* [34] s výstupy generovanými aplikací, které byly vytvořeny na základě výběrové transformace podle různých os a prahových hodnot.

Originální obraz (obrázek č. 22) představuje čtyři různé případy transformací části dat. V každém ze čtverců je aplikována transformace na jinou podmnožinu generovaných bodů. Tyto body mají charakter rozptýlených značek „+“ a výsledkem je vizuálně rozpoznatelná deformace části původního rozložení, zatímco zbytek dat zůstává v původní podobě. Každá z těchto částí ukazuje jiný segment transformační roviny – například levý spodní roh, pravý horní atd. – přičemž transformace mění tvar oblasti do zakřivené nebo stažené formy, aniž by se dotkla zbývajících bodů.



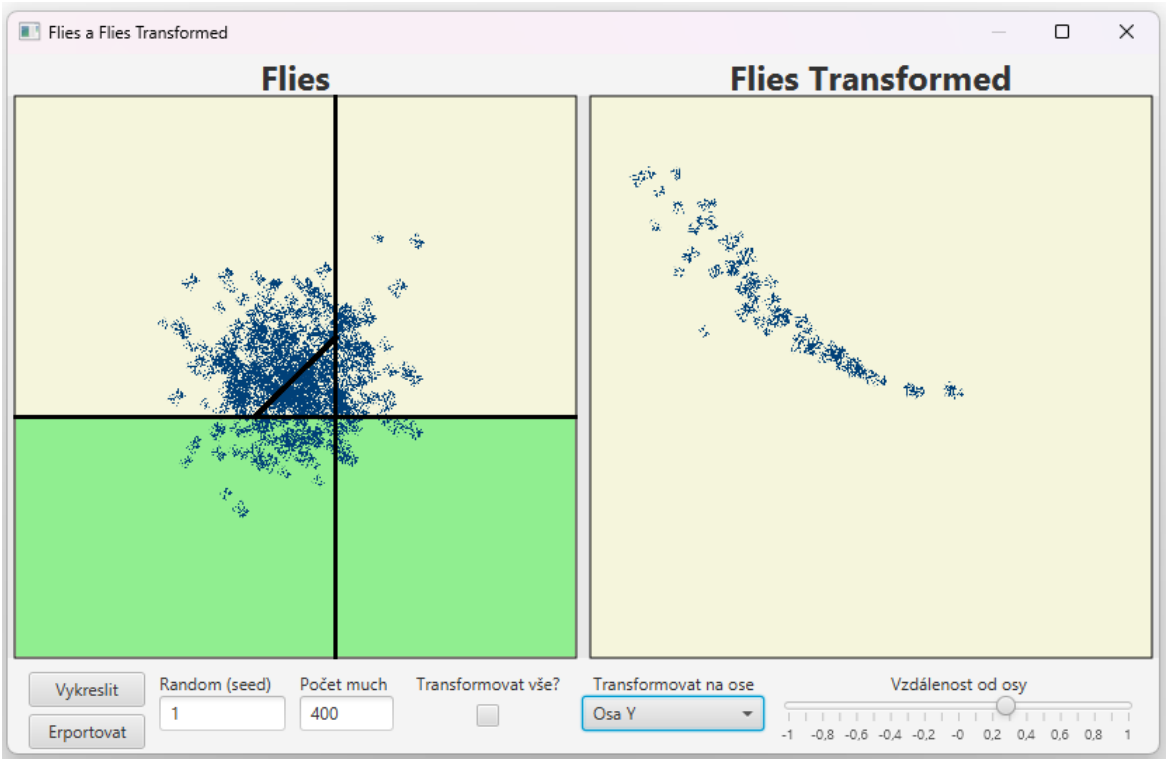
Obrázek 22 – Flies Transformed [34]

V aplikaci byla obdobná logika replikována moderním způsobem – místo bodů jsou vykreslovány mušky a výběr oblasti, která má být transformována, se provádí interaktivně volbou osy (X nebo Y) a nastavením prahové hodnoty posuvníkem. V obrázku č. 23 je zvolen výběr levé poloviny vzhledem k ose X. Transformace je aplikována pouze na mušky, které se nacházejí na levé straně zeleně zvýrazněného pole. Výsledkem je, že pravá část obrazu (v pravém panelu) zobrazuje transformované mušky vytažené do úzké zakřivené formace, velmi podobné levému dolnímu čtverci originálu. Obrázek č. 24 ukazuje opačný případ, kde jsou transformovány mušky na pravé straně pole, a výsledkem je zakřivená formace směřující opačně, podobná pravému hornímu kvadrantu z originálu. Obrázek č. 25 demonstruje výběr horní poloviny mušek podle osy Y. Výsledná transformace vytváří zakřivený útvar v pravém horním směru, který odpovídá deformaci dat z horní části původního kruhu. Obrázek č. 26 pak ukazuje transformaci spodní poloviny, což vede k podobnému efektu, jako má pravý dolní kvadrant originálního obrázku.

Obrázek 23 – Aplikace Flies a Flies Transformed s výběrem necelé levé poloviny vzhledem k ose X

Obrázek 24 – Aplikace Flies a Flies Transformed s výběrem necelé pravé poloviny vzhledem k ose X

Obrázek 25 – Aplikace Flies a Flies Transformed s výběrem necelé vrchní poloviny vzhledem k ose Y



Obrázek 26 – Aplikace Flies a Flies Transformed s výběrem necelé spodní poloviny vzhledem k ose Y

Při srovnání obou dvojic – původního a nového *Flies* i *Flies Transformed* – lze říci, že aplikace věrně zachycuje principy původního generativního algoritmu Charlese Csuriho, ale zároveň je rozšiřuje o moderní technické prostředky, vyšší rozlišení, větší počet prvků a preciznější vizualizaci. Výsledný dojem z výstupů je výtvarně konzistentní s originálem, přestože technické zpracování je pochopitelně současné. Důležité je, že aplikace zachovává klíčové koncepty: náhodnost s řízeným rozložením, individuální variabilitu prvků a možnost matematické transformace prostoru jako prostředek estetické manipulace. Tím dochází nejen k rekonstrukci historického přístupu, ale i k jeho interpretaci v kontextu současného vizuálního programování.

Tato kontinuita s původními díly je dále podpořena implementací selektivní transformace, která se vizuálně i logicky přibližuje původním experimentům s částečnou deformací obrazových dat. Aplikace zde opět přesně napodobuje principy navržené v původním výzkumu – uživatel si může interaktivně zvolit osu a hodnotu, podle níž se rozhoduje, která část obrazového pole bude transformována, a výsledek je okamžitě vizuálně zobrazen. Díky tomu má uživatel k dispozici přímou vizuální zpětnou vazbu prostřednictvím zeleně zvýrazněné oblasti a referenční čáry, která zřetelně ukazuje hranici výběru. Na rozdíl od původních statických výstupů zde vzniká dynamický systém, ve kterém lze parametry upravovat a sledovat jejich účinek v reálném čase. Vizuálně a strukturálně tak moderní výstupy přesně odpovídají historickým předlohám, ale zároveň přinášejí mnohem větší flexibilitu, přesnost a možnost zpětné kontroly.

Aplikace tím přerůstá roli pouhé rekonstrukce a stává se nástrojem nejen pro tvorbu generativního umění, ale i pro jeho studium, analýzu a výukové využití, zejména při vysvětlování nelineárních transformací v souřadnicových systémech. Její přínos spočívá nejen ve schopnosti věrně reprodukovat historický výsledek, ale i v rozšíření možností experimentace a pochopení principů, které za těmito díly stály.

2.3.1 Random War of Soldiers

Analýza originálního díla

Dílo *Random War* od Charlese Csuriho a Jamese Shaffera, vytvořené v roce 1967, patří mezi vizuálně i konceptuálně nejkomplexnější příklady raného počítačového umění, které experimentuje s kombinací náhodnosti, pravidelné struktury a významové symboliky. Obraz je rozdělen na dvě hlavní části: horní typografickou oblast obsahující seznamy jmen, a spodní dynamické pole plné figurálních forem připomínajících vojáky v různých pozicích. Celé dílo působí jako syntéza datové vizualizace, výtvarné struktury a konceptuálního komentáře k absurditě války.

Horní část obrazu zobrazuje uspořádané sloupce jmen rozdělené do kategorií jako „DEAD“, „WOUNDED“, „MISSING“, „SURVIVING“ a „MEDALS AWARDED“. Tato jména – formátovaná strojopisem – jsou jména reálných Csuriho spolubojovníků z bitvy v Ardenách, svou formou připomínají vojenské záznamy nebo oficiální výpisy z databází padlých a přeživších. Přesná typografie, opakující se zkratky hodností a sériová čísla umocňují dojem, že se nejedná jen o vizuální prvek, ale o datově řízený výstup. Vztah mezi těmito jmény a

spodní figurální scénou není explicitně mechanický, ale je tematicky provázaný: horní část zastupuje statistiku a evidenci, zatímco spodní chaos a tělesnost samotného konfliktu.

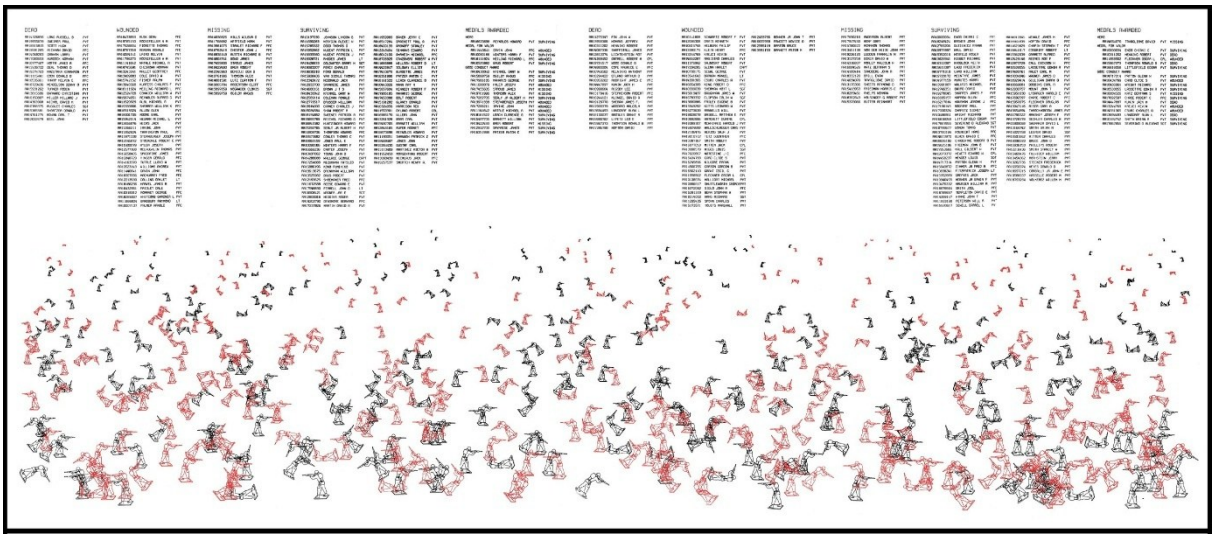
Ve spodní části díla se rozprostírá pole zaplněné množstvím figur v černé a červené barvě. Tyto figury, kreslené pomocí jednoduchých čar, představují vojáky v různých pózách – někteří útočí, jiní klesají, padají nebo se míjejí. Barvy a překrývání vytvářejí dojem pohybu a intenzity. Některé postavy se opakují na stejném místě ve více vrstvách, čímž vzniká efekt nahromadění jakoby opakovaného záznamu pohybu nebo hromadného pádu. Náhodné rozložení figur a jejich hustota není rovnoměrná – některé oblasti jsou prázdnější, jiné přeplněné – čímž obraz získává dramatický rytmus a napětí.

Za tímto chaosem se však skrývá jasně definovaný výpočetní princip. Každá figura je generována jako vektorová kompozice podle pevně stanoveného repertoáru póz a směrů. Algoritmus náhodně určuje nejen jejich typ, ale i jejich pozici v prostoru a jejich vztah k okolním postavám. Pomocí řízené náhodnosti jsou v některých oblastech formovány menší shluky, zatímco jinde zůstávají jednotlivci izolovaní. Tato pravděpodobnostní distribuce připomíná simulaci bitevního pole, kde dochází k lokálním střetům i prázdným zónám.

Významovým jádrem *Random War* je právě napětí mezi strukturou a chaosem, mezi čitelností dat a vizuální explozí. Seznamy jmen nahoře vytvářejí rámec – mohou být chápány jako vstupní data nebo jako důsledky vizuálně zobrazené situace dole. Celý obraz tak funguje jako vizuální metafora války jako statistického a zároveň nevyzpytatelného dění. Všechny prvky byly generovány pomocí počítačového programu, který kombinoval algoritmické rozhodování s náhodností, a výstup byl následně převeden na kresbu plotterem.

Výsledné dílo tak nepředstavuje jen výtvarnou kompozici, ale také datafikovaný model konfliktu, ve kterém je každý voják údajem a zároveň obrazem. *Random War* spojuje výpočetní estetiku s morálním a politickým podtextem, čímž se stává nadčasovým příkladem generativního umění, které přesahuje čistě formální experiment a vstupuje do oblasti vizuální eseje o válce, evidenci, osudu a paměti.

„A drawing was made of one toy soldier and this became the data deck. A computer program which generates random numbers is called a pseudo-random number generator. Such a program determined the distribution and position of 400 soldiers on the battlefield. One side is called the "Red" and the other one the "Black", and the names of real people were given to the program. Another computer program assigned military ranks and army serial numbers at random. The random number generator decided the following information and the computer made this picture with the casualty list. (1) Dead (2) Wounded (3) Missing (4) Survivors (5) One Hero for each side (6) Medals for Valor (7) Good Conduct (8) Efficiency Medals.“ – Charles Csuri [29].



Obrázek 27 – Random War [23]

I přes to že sám Charles Csuri uvádí že bylo vygenerováno 400 vojáků, kde každá strana měla 200 vojáků, tak po mé detailní analýze, jsem zjistil že tomu minimálně v rámci počtu jmen není. I přes to že se celkový součet blíží k 200 na každé straně, tak je důležité to že některé jména vojáků, co byly vyznamenání medailí jsou uvedeni jak v medailích, tak ve sloupci s jejich stavem, a někteří jsou pouze ve sloupci s medailemi. Počty z originálního obrazu jsou zaneseny v následující tabulce.

Tabulka 1 – Počet vojáků, jejich stavů a medailí

Strana	Dead	Wounded	Missing	Survivors	Medals	Hero	Medals for Valor	Good Conduct Award
RED	20	40	14	70	20	1	4	15
BLACK	22	44	16	74	18	1	5	12

Struktura a inicializace scény

Struktura a inicializace scény v části aplikace *Random War* tvoří základní rámec pro následné vykreslování simulace. Po spuštění této části programu dojde automaticky k načtení FXML rozhraní, které definuje uspořádání jednotlivých komponent – výkresového prostoru, panelu se statistikami, textových polí pro vstupy a ovládacích prvků, jako jsou tlačítka, přepínače a posuvníky. V okamžiku, kdy je scéna plně načtena, je zavolána metoda initialize(), která zajišťuje, že jsou všechny ovládací komponenty propojeny s logikou programu a že mají správné výchozí nastavení.

Součástí inicializace je nastavení výchozích hodnot vstupních polí, například přednastavení hodnot rozdělení statusů (27 % pro „dead“, 12 % pro „wounded“, 6 % pro „missing“ a 55 %

pro „surviving“) nebo počtu vojáků. Zároveň se připraví výběrové komponenty, jako jsou volby stran nebo typ písma pro seznam jmen, a nastaví se implicitní volby. Kromě toho se registrují posluchače událostí, které reagují na změny velikosti výkresového panelu. To znamená, že při každé změně šířky nebo výšky výkresu se automaticky překreslí obsah, aby odpovídal aktuálním rozměrům okna. Stejně tak je napojen posuvník velikosti písma v seznamu statistik, takže jakákoli změna velikosti fontu se okamžitě promítne do vzhledu textového panelu.

Metoda pro nastavení výchozích hodnost statusů a medailí vypadá takto:

```
// Nastavení základních poměrů
private void initDefaultRatios() {
    for (Side side : Side.values()) {
        // Poměry stavů
        statusRatios.putIfAbsent(side, Map.of(
                Status.SURVIVING,0.55,
                Status.MISSING,0.06,
                Status.WOUNDED,0.12,
                Status.DEAD,0.27

        ));
        // Poměry medailí
        medalRatios.putIfAbsent(side, Map.of(
                Honor.VALOR,0.04,
                Honor.GOOD_CONDUCT,0.15
        ));
    }
}
```

Inicializace také zajišťuje přípravu obrazových a datových struktur, do kterých budou ukládáni jednotliví vojáci, jejich atributy a pozice. Tyto datové struktury jsou vymazány a připraveny vždy před novým vykreslením, aby se zabránilo přenášení dat mezi jednotlivými simulacemi. Významným krokem je načtení seznamu jmen ze souboru, který se provádí pouze jednou během inicializace a poté je použit při generování vojáků.

Tím, že jsou všechny komponenty zaintegrovány ještě před prvním vstupem uživatele, je zajištěno, že program je připraven k okamžitému použití. Veškeré vazby mezi rozhraním a datovou logikou jsou navázány, vstupy mají validní hodnoty a scéna je schopna okamžitě zareagovat na pokyn ke generování simulace. Výsledkem této důsledné inicializace je plynulý, robustní a předvídatelný chod celé aplikace.

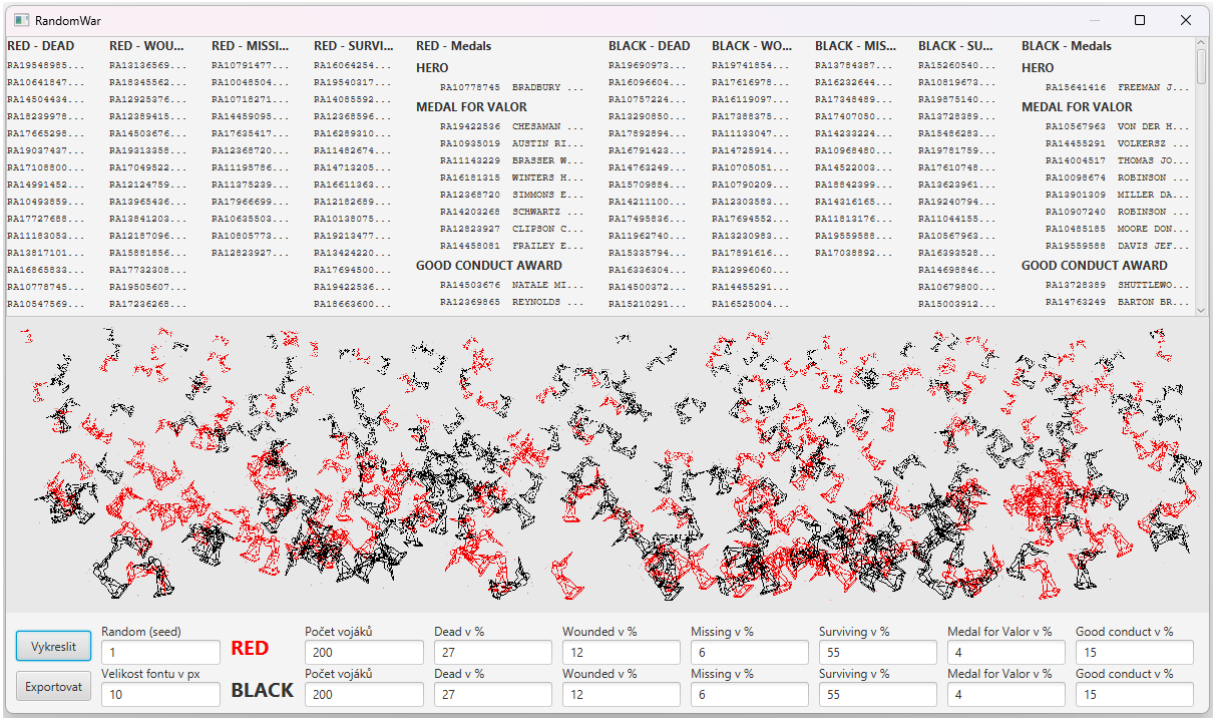
Metoda inicializace programu vypadá takto:

```
public void initialize(URL url, ResourceBundle resourceBundle) {
    // Při změně velikosti panelů překresli GUI
    root.widthProperty().addListener(cl -> redraw());
    root.heightProperty().addListener(cl -> redraw());
    pane.widthProperty().addListener(cl -> redraw());
    pane.heightProperty().addListener(cl -> redraw());

    // HBox zabere 60 % výšky hlavního kontejneru
    hBox.prefHeightProperty().bind(root.heightProperty().multiply(0.6));

    // Výška pane: max 40 % výšky root nebo 5:16 poměr podle šířky
```

```
        pane.prefHeightProperty().bind(
                Bindings.min(
                        root.heightProperty().multiply(0.4),
                        pane.widthProperty().multiply(5.0/16.0)
                )
        );

        // Inicializace výchozích poměrů a načtení jmen
        initDefaultRatios();
        loadNameList();

        // Spustit vykreslení až po inicializaci GUI (jinak null velikosti)
        Platform.runLater(this::redraw);
    }
```

Uživatelské vstupy a jejich validace

Uživatelské vstupy v části aplikace *Random War* představují klíčový prvek, který umožňuje ovlivnit výslednou podobu simulace a zároveň zajišťuje její interaktivní a variabilní charakter. Uživatel má možnost zadat celou řadu parametrů, které ovlivňují jak datový základ (například počet vojáků nebo pravděpodobnost výskytu jednotlivých statusů), tak vzhled a chování vizualizace (například velikost písma nebo typ medailí). Tato zadání se provádějí prostřednictvím textových polí, posuvníků, zaškrtávacích polí a výběrových seznamů v grafickém rozhraní.



Obrázek 28 – Základní zobrazení aplikace Random War (v dolní části jsou uživatelské vstupy)

Při zadávání hodnot je klíčové, aby aplikace ověřila, zda zadané vstupy dávají smysl a jsou ve správném formátu. Tento proces validace je v aplikaci důkladně ošetřen. Veškeré vstupy z textových polí jsou převáděny na číselné hodnoty v rámci bloku *try-catch*, který umožňuje zachytit chyby typu *NumberFormatException*. Pokud například uživatel zadá písmeno nebo speciální znak namísto čísla, aplikace nezkolabuje, ale zobrazí uživatelsky přívětivou chybovou hlášku s výzvou k nápravě. Stejně tak je kontrolováno, že pole nejsou prázdná, že hodnoty nejsou záporné a že celkové procentuální rozdělení statusů nebo medailí dává dohromady logický součet.

Program rovněž kontroluje specifické podmínky, které by mohly způsobit nekonzistentní chování. Například se ověřuje, že obě strany mají přiděleno více než 0 vojáků, že výsledná velikost písma v seznamu statistik není menší než 1, a že poměrné rozdělení medailí nevede k nulovému výběru všech kategorií. Pokud jsou porušeny tyto nebo jiné vnitřní podmínky, je uživatel upozorněn prostřednictvím modálního dialogu, v němž je jasně popsán charakter chyby. Výsledkem je, že do další fáze výpočtu a vykreslení se dostanou pouze validní a bezpečné hodnoty.

Díky tomuto systému ochrany vstupů je aplikace stabilní a odolná vůči chybám způsobeným nesprávným zadáním, a zároveň zůstává uživatelsky vstřícná, protože každé pochybení je ihned oznamováno srozumitelným způsobem. Validace tak tvoří důležitý most mezi flexibilitou uživatelského zadání a integritou výpočetní logiky simulace.

Kód pro validace a využití uživatelských na zadávání poměrů statusů a medailí vypadá takto:

```
private void applyRatiosAndRedraw() {
    try {
        // Přečteme RED procenta z UI a převedeme na poměr 0.0–1.0
(percentage / 100)
        double deadR = Double.parseDouble(tfDeadRed.getText()) / 100.0;
        double woundR = Double.parseDouble(tfWoundedRed.getText()) / 100.0;
        double missR = Double.parseDouble(tfMissingRed.getText()) / 100.0;
        double survR = Double.parseDouble(tfSurvivingRed.getText()) /
100.0;

        // Negativní kontrola pro RED stavy
        if (deadR < 0 || woundR < 0 || missR < 0 || survR < 0) {
            showError("Chyba vstupu", "Negativní procento", "Procenta stavů
pro RED nesmí být záporná.");
            return;
        }

        // Kontrola, zda Red stavy dohromady dávají 100 %
        double sumRed = deadR + woundR + missR + survR;
        if (Math.abs(sumRed - 1.0) > 1e-6) {
            showError("Chyba poměrů", "Poměry stavů pro RED nejsou platné",
                    String.format("Součet procent musí být 100%%, aktuálně
je %.2f%%", sumRed * 100));
            return;
        }
        statusRatios.put(Side.RED, Map.of(
                Status.DEAD,      deadR,
                Status.WOUNDED,   woundR,
                Status.MISSING,   missR,
                Status.SURVIVING, survR
```

```
        ));

        // Přečteme RED procenta z UI a převedeme na poměr 0.0–1.0
(percentage / 100)
        double valorR = Double.parseDouble(tfValorRed.getText()) / 100.0;
        double goodR  = Double.parseDouble(tfConductRed.getText()) / 100.0;

        // Negativní kontrola pro RED medaili
        if (valorR < 0 || goodR < 0) {
            showError("Chyba vstupu", "Negativní procento", "Procenta
medailí pro RED nesmí být záporná.");
            return;
        }

        // Kontrola, zda Red medaile dohromady nepřesahují 100 %
        double sumMedRed = valorR + goodR;
        if (sumMedRed > 1.0 + 1e-6) {
            showError("Chyba poměrů", "Poměry medailí pro RED překračují
100 %",
                    String.format("Součet procent nesmí být nad 100%%,
aktuálně je %.2f%%", sumMedRed * 100));
            return;
        }
        medalRatios.put(Side.RED, Map.of(
                Honor.VALOR,        valorR,
                Honor.GOOD_CONDUCT, goodR
        ));

        // Přečteme BLACK procenta z UI a převedeme na poměr 0.0–1.0
(percentage / 100)
        double deadB = Double.parseDouble(tfDeadBlack.getText()) / 100.0;
        double woundB = Double.parseDouble(tfWoundedBlack.getText()) /
100.0;
        double missB = Double.parseDouble(tfMissingBlack.getText()) /
100.0;
        double survB = Double.parseDouble(tfSurvivingBlack.getText()) /
100.0;

        // Negativní kontrola pro BLACK stavy
        if (deadB < 0 || woundB < 0 || missB < 0 || survB < 0) {
            showError("Chyba vstupu", "Negativní procento", "Procenta stavů
pro BLACK nesmí být záporná.");
            return;
        }

        // Kontrola, zda BLACK stavy dohromady dávají 100 %
        double sumBlack = deadB + woundB + missB + survB;
        if (Math.abs(sumBlack - 1.0) > 1e-6) {
            showError("Chyba poměrů", "Poměry stavů pro BLACK nejsou
platné",
                    String.format("Součet procent musí být 100%%, aktuálně
je %.2f%%", sumBlack * 100));
            return;
        }
        statusRatios.put(Side.BLACK, Map.of(
                Status.DEAD,      deadB,
                Status.WOUNDED,   woundB,
                Status.MISSING,   missB,
                Status.SURVIVING, survB
        ));
```

```
        // Přečteme BLACK procenta z UI a převedeme na poměr 0.0–1.0
(percentage / 100)
        double valorB = Double.parseDouble(tfValorBlack.getText()) / 100.0;
        double goodB  = Double.parseDouble(tfConductBlack.getText()) /
100.0;

        // Negativní kontrola pro BLACK medaili
        if (valorB < 0 || goodB < 0) {
            showError("Chyba vstupu", "Negativní procento", "Procenta
medailí pro BLACK nesmí být záporná.");
            return;
        }

        // Kontrola, zda BLACK medaile dohromady nepřesahují 100 %
        double sumMedBlack = valorB + goodB;
        if (sumMedBlack > 1.0 + 1e-6) {
            showError("Chyba poměrů", "Poměry medailí pro BLACK překračují
100 %",
                    String.format("Součet procent nesmí být nad 100%%,
aktuálně je %.2f%%", sumMedBlack * 100));
            return;
        }
        medalRatios.put(Side.BLACK, Map.of(
                Honor.VALOR,        valorB,
                Honor.GOOD_CONDUCT, goodB
        ));

        // Kontrola velikosti fontu, nesmí být menší než 1
        fontSize = Integer.parseInt(tfFontSize.getText());
        if (fontSize < 1){
            showError("Chyba velikosti fontu", "Příliš malý font",
"Velikost fontu musí být větší než 1");
            return;
        }

        // Vše v pořádku – aktualizuj zobrazení
        redraw();
    } catch (NumberFormatException ex) {
        showError("Chybný vstup", "Neplatné číslo", "Zadejte platné
procento (0-100). " + ex.getMessage());
    }
}
```

Generování pseudonáhodných prvků

Generování pseudonáhodných čísel v aplikaci *Random War* tvoří základní mechanismus, na kterém stojí celá logika náhodné distribuce vlastností jednotlivých vojáků. Použit je přitom standardní generátor pseudonáhodných čísel třídy *Random* ze standardní knihovny Javy. Tento generátor je deterministický, což znamená, že pro stejné počáteční nastavení – tzv. *seed* – bude generovat vždy stejnou posloupnost náhodných hodnot. Tato vlastnost je v aplikaci plně využita, protože uživatel má možnost seed zadat ručně, čímž je zajištěna opakovatelnost simulace. Pokud uživatel zadá stejný seed a stejné parametry, výstup bude vždy totožný. Tento princip umožňuje snadné porovnávání různých variant nebo prezentaci konkrétní konfigurace.

Každý voják je vykreslen jako grafický objekt reprezentující stylizovanou postavu. Aby výstup nepůsobil mechanicky nebo předvídatelně, je poloha každého vojáka určena pomocí pseudonáhodných čísel. Tyto pozice jsou generovány samostatně pro osu X a osu Y v rámci výkresové plochy. Pro každého vojáka se náhodně vygeneruje relativní hodnota v rozmezí 0–1, která je následně vynásobena aktuální šířkou a výškou výkresové oblasti. Díky tomu je zajištěno, že se postavy rozprostřou rovnoměrně po celé scéně bez kumulace v konkrétních částech.

Po výpočtu absolutních souřadnic je pozice jemně upravena tak, aby voják nepřesahoval okraje výkresu. Program počítá s tím, že každá postava má určitou šířku a výšku, takže při umístění na plátno je třeba zajistit, že její střed zůstane dostatečně vzdálen od okrajů – například minimálně o polovinu výšky postavy. Tím se zamezuje tomu, aby byl některý voják oříznut nebo aby vizuální kompozice působila nevyváženě. Tato ochrana hranic se provádí výpočtem korekčního offsetu, který je přičten nebo odečten podle velikosti plochy.

Vedle pozice je náhodně určováno také natočení každého vojáka. K tomu dochází výpočtem náhodné hodnoty úhlu v rozsahu přibližně od −85° do +85°, což zajišťuje různorodost směru postavy bez úplného převrácení. Kromě toho může být každá postava navíc horizontálně zrcadlena – tato operace je řízena náhodným booleovským rozhodnutím, které určí, zda bude textura překlopena kolem svislé osy. Kombinací těchto dvou parametrů – rotace a zrcadlení – vzniká velmi široké spektrum možných vizuálních variant jediné základní textury.

Zásadním vizuálním prvkem je také proměnlivá velikost vojáků v závislosti na jejich vertikální pozici. Tento efekt simuluje perspektivu a hloubku prostoru, čímž výstup působí prostorově a organicky. V praxi to znamená, že vojáci umístění blíže hornímu okraji výkresu (tedy v „dálce“) jsou zmenšeni, zatímco ti blíže dolnímu okraji („vpředu“) jsou větší. Tento efekt je vypočítán lineární funkcí, kde Y-ová pozice určuje škálovací faktor. Například postava ve spodní třetině výkresu může být zobrazena na 100 % velikosti, zatímco postava v horní třetině pouze na 40 %. Výsledná velikost ovlivňuje nejen výšku a šířku obrazového prvku, ale i oblast, ve které může být umístěn, protože zmenšení či zvětšení mění riziko kolize s okraji výkresu.

Kód pro vygenerovaní relativní pozice vojáka, přepočtu na reálnou pozici, výpočet velikosti a vygenerování rotace vypadá takto:

```
 private Rectangle createRectangle(Soldier soldier, double width, double
height){
    // Minimální a maximální velikost vojáka podle šířky
    double minSize = 20 / (2200 / width);
    double maxSize = 80 / (2200 / width);

    // Rozsah pro náhodné umístění
    double boundX = width  - (maxSize);
    double boundY = height - (20 + maxSize);
    if (boundX <= 0) boundX = 1;
    if (boundY <= 0) boundY = 1;

    // Náhodná pozice
    double x = random.nextDouble(boundX);
    double y = 10 + random.nextDouble(boundY);
```

```
    // Velikost vojáka závisí na svislé pozici
    double relativePosition = (y / height);
    double soldierWidth = minSize + relativePosition * (maxSize - minSize);
    double soldierHeight = soldierWidth * (80.3 / 73.0); // Poměr obrázku

    // Vytvoření obdélníku s texturou podle strany
    Rectangle rectangle = new Rectangle(x, y, soldierWidth, soldierHeight);
    rectangle.setFill(soldier.getSide() ==
Side.RED?imagePatternRed:imagePatternBlack);

    // Náhodné zrcadlení obrázku
    if(random.nextBoolean()){
        rectangle.setScaleX(-1);
        rectangle.setX(rectangle.getX()+soldierWidth);
    }

    // Náhodná rotace
    rectangle.setRotate(-85 + random.nextInt(170));

    return rectangle;
}
```

Jedním z důležitých atributů každého vojáka je jeho status, tedy údaj o jeho osudu nebo stavu. Uživatel zadává v rozhraní poměrové rozdělení pro čtyři kategorie: *dead* (mrtvý), *wounded* (zraněný), *missing* (nezvěstný) a *surviving* (přeživší). Tyto poměry jsou zadávány v procentech a aplikace provádí jejich validaci – kontroluje, že součet všech čtyř hodnot je přesně 100 % a že jednotlivé položky nejsou záporné. Po potvrzení vstupů program převede procenta na absolutní počty vojáků pro každou kategorii. Výběr konkrétních jedinců do těchto skupin pak probíhá čistě náhodně. Program udržuje seznam dosud nepřiřazených vojáků a pro každou kategorii náhodně vybírá, dokud není vyčerpán požadovaný počet. Tím je zajištěno, že každý voják získá právě jeden status a zároveň že jsou poměry přesně dodrženy. Pokud dojde k zaokrouhlovací odchylce, rozdíl je korigován přidáním zbývajících vojáků do poslední skupiny.

Metoda pro přiřazení statusů všem vojákům podle nastavených poměrů vypadá takto:

```
private void assignStatuses(List<Soldier> list, Map<Status,Double> ratios){
    int total = list.size(); // Počet vojáků k přiřazení stavů

    // Spočítáme kolik vojáků má mít každý stav (zaokrouhleno dolů)
    Map<Status,Integer> count = new LinkedHashMap<>();
    ratios.forEach((s, r) -> count.put(s, (int) Math.floor(r * total))
    );

    // Kolik vojáků už má přiřazený stav?
    int assigned = count.values().stream().mapToInt(i -> i).sum();
    int left = total - assigned; // Kolik jich ještě zbývá?

    // Rozdělení zbylých (např. kvůli zaokrouhlování) mezi jednotlivé stavy
    Iterator<Status> it = count.keySet().iterator();
    while (left-- > 0 && it.hasNext()) {
        Status s = it.next();
        count.put(s, count.get(s) + 1);
    }

    // Vytvoříme seznam statusů podle počtu – např. [SURVIVING, SURVIVING,
```

```
DEAD,...]
    List<Status> pool = count.entrySet().stream()
            .flatMap(e -> Collections.nCopies(e.getValue(),
e.getKey()).stream()).collect(Collectors.toList());

    Collections.shuffle(pool, random); // Zamícháme, aby nebyly stejné
stavy vedle sebe

    // Každému vojákovi přiřadíme jeden stav z poolu
    for (int i = 0; i < total; i++) {
        list.get(i).setStatus(pool.get(i));
    }
}
```

Podobným způsobem funguje také přidělování medailí, které představují další vrstvu náhodné distribuce. Uživatel má možnost nastavit poměr dvou typů vyznamenání: *Medal for Valor* a *Good Conduct Award*, pro každou armádu zvlášť. Stejně jako u statusů probíhá validace poměrů a převod na konkrétní počty. Přidělení medailí pak probíhá opět náhodně – z výběru přeživších vojáků se vybírá podmnožina, které jsou přiřazeny vyznamenání podle daných kvót. Aplikace zároveň zajišťuje, že právě jeden voják na každé straně obdrží medaili *Hero*. Toto zajišťuje, že každá simulace bude obsahovat klíčový symbolický prvek hrdinství, a zároveň se tím rozšiřuje narativní vrstva výstupu.

Metoda pro přiřazení medailí všem vojákům podle nastavených poměrů vypadá takto:

```
private void assignMedals(List<Soldier> list, Map<Honor, Double> ratios) {
    int total = list.size();

    // Spočítáme kolik vojáků má dostat každou medaili
    Map<Honor, Integer> counts = new LinkedHashMap<>();
    ratios.forEach((m, r) -> counts.put(m, (int) Math.floor(r * total))
    );

    // Zaručíme, že vždy někdo dostane HERO (pokud nebyl zadán)
    counts.putIfAbsent(Honor.HERO, 1);

    // Zbytek vojáků bude bez medaile (HONOR.NONE)
    int assigned = counts.values().stream().mapToInt(i -> i).sum();
    int noneCount = total - assigned;
    if (noneCount < 0) noneCount = 0;
    counts.put(Honor.NONE, noneCount);

    // Vytvoříme pool medailí (např. [VALOR, GOOD_CONDUCT, NONE, ...])
    List<Honor> pool = counts.entrySet().stream()
            .flatMap(e -> Collections.nCopies(e.getValue(),
e.getKey()).stream()).collect(Collectors.toList());

    Collections.shuffle(pool, random);  // Zamícháme, aby nebyly stejné
medaile vedle sebe

    // Každému vojákovi přiřadíme medaili
    for (int i = 0; i < total; i++) {
        list.get(i).setHonor(pool.get(i));
    }
}
```

Celý proces distribuce statusů a medailí je pevně svázán s centrálním generátorem náhodných čísel a probíhá vždy podle stejné logiky: vstupní parametry → přepočet poměrů → náhodný výběr. Tím je zajištěna konzistence, ale i vizuální a datová variabilita mezi simulacemi.

Výsledkem tohoto komplexního systému je obraz, který nejen působí živě a nepředvídatelně, ale zároveň přesně odráží zadanou konfiguraci a respektuje pravidla, která uživatel stanovil. Kombinace centrálně řízeného generátoru pseudonáhodných čísel a pevně definovaných parametrů umožňuje dosáhnout rovnováhy mezi náhodností a opakovatelností, čímž se vytváří výstup, který je vizuálně rozmanitý, a přesto zcela reprodukovatelný. Tento přístup umožňuje propojit sílu náhodného výběru s přesnou kontrolou nad jednotlivými aspekty simulace, což přímo navazuje na filozofii původního díla *Random War* – tedy na paradox náhodně určeného, ale strukturálně determinovaného výsledku.

Systém spojení náhodného rozložení, směrového natočení a perspektivního škálování navíc zajišťuje, že výstupní obraz nepůsobí jako pravidelná mřížka nebo opakující se vzor, ale jako dynamické a prostorově věrohodné pole pohybujících se postav. Vizuální variabilita je dosažena nejen náhodnou pozicí, ale také jemně odlišným úhlem rotace a zrcadlením, což eliminuje uniformitu a dodává scéně výrazný rytmus. Proměnlivá velikost vojáků podle jejich svislé pozice zároveň simuluje hloubku prostoru a vtahuje diváka do iluzivní perspektivy, v níž jsou jednotlivé postavy vnímány jako vzdálené či blízké. Každý voják se tak stává unikátní instancí, která zároveň odpovídá výpočetnímu modelu i výtvarnému záměru. Výsledný efekt je nejen esteticky přesvědčivý, ale i koncepčně silný – náhodně generovaný, ale kompozičně stabilní obraz konfliktu, který svým prostorovým uspořádáním podtrhuje symbolickou rovinu algoritmické války.

Interaktivita ve vizualizaci

Interaktivita ve vizualizaci aplikace *Random War* hraje důležitou roli při propojování obrazové a datové složky simulace a výrazně přispívá k celkovému uživatelskému zážitku. Uživatel není pouze pasivním pozorovatelem výsledného obrazu, ale může s ním aktivně pracovat a získávat detailní informace o každém jednotlivém prvku ve scéně. Každý voják, který je vykreslen na výkresové ploše, má totiž svou textovou reprezentaci v přehledném seznamu statistik, který obsahuje jméno, hodnost, status a případné ocenění.

Metoda pro vyplnění jmen, čísel, stavů a medailí do vrchní části obrazu vypadá takto:

```
private void populateStatsBox() {
    hBox.getChildren().clear(); // Vyčisti pravý panel se statistikami

    for (Side side : Side.values()) {
        for (Status st : Status.values()) {
            VBox col = new VBox(5); // Jeden sloupec pro konkrétní stav
            Label lblH = new Label(side + " - " + st); // Nadpis sloupce
            lblH.setFont(Font.font("System", FontWeight.BOLD, (fontSize +
2)));
            col.getChildren().add(lblH);

            // Vyfiltruj vojáky dané strany a stavu
            soldiers.stream()
                    .filter(v->v.getSide()==side && v.getStatus()==st)
                    .map(v-> {
```

```
                        String key = v.getPersonalNumber();
                        String text = String.format("%s  %-20s %-3s", key,
v.getName(), v.getRank());
                        Label lbl = new Label(text);
                        lbl.setFont(Font.font("Monospaced", fontSize));

                        // Ulož label pro případné zvýraznění
                        soldierLabels.computeIfAbsent(key, k->new
ArrayList<>()).add(lbl);

                        // Přidání reakce na myš
                        lbl.setOnMouseEntered(e -> {
                            Rectangle r = soldierRects.get(key);
                            if(r!=null) highlightRect(r,key,true);
                        });
                        lbl.setOnMouseExited(e -> {
                            Rectangle r = soldierRects.get(key);
                            if(r!=null) highlightRect(r,key,false);
                        });
                        return lbl;
                    })
                    .forEach(col.getChildren()::add);

            hBox.getChildren().add(col); // Přidej sloupec do hlavního
panelu
        }

        // Vytvoř sloupec s medailemi pro tuto stranu
        VBox medalCol = new VBox(5);
        Label lblH = new Label(side + " - Medals"); // Nadpis sloupce
        lblH.setFont(Font.font("System", FontWeight.BOLD, (fontSize + 2)));
        medalCol.getChildren().add(lblH);

        for (Honor h : Honor.values()) {
            if (h==Honor.NONE) continue;
            Label lblH2 = new Label(h.toString()); // Podnadpis sloupce
            lblH2.setFont(Font.font("System", FontWeight.BOLD, (fontSize +
2)));
            medalCol.getChildren().add(lblH2);

            // Vyfiltruj vojáky dané strany a medaile
            soldiers.stream()
                    .filter(v->v.getSide()==side && v.getHonor()==h)
                    .map(v-> {
                        String key = v.getPersonalNumber();
                        String text = String.format("    %s  %-20s %-3s
%s", key, v.getName(), v.getRank(), v.getStatus());
                        Label lbl = new Label(text);
                        lbl.setFont(Font.font("Monospaced", fontSize));

                        // Ulož label pro případné zvýraznění
                        soldierLabels.computeIfAbsent(key, k->new
ArrayList<>()).add(lbl);

                        // Přidání reakce na myš
                        lbl.setOnMouseEntered(e -> {
                            Rectangle r = soldierRects.get(key);
                            if(r!=null) highlightRect(r,key,true);
                        });
                        lbl.setOnMouseExited(e -> {
                            Rectangle r = soldierRects.get(key);
```

```
                            if(r!=null) highlightRect(r,key,false);
                        });
                        return lbl;
                    })
                    .forEach(medalCol.getChildren()::add);
        }

        hBox.getChildren().add(medalCol); // Přidej sloupec s medailemi
    }

    statsScroll.setFitToWidth(true); // Šířka sloupců přizpůsobena
scrollovací oblasti
}
```

Mezi vizuální a textovou částí existuje přímá vazba – jakmile uživatel myší přejede nad kteroukoli postavou v grafické části, odpovídající řádek v textovém seznamu se automaticky zvýrazní. Totéž platí i opačně: pokud kurzor přejede nad záznamem ve výpisu statistik, zvýrazní se odpovídající postava na výkresu. Zvýraznění je realizováno vizuálními efekty, jako je barevné podbarvení, rámeček nebo optická záře. Díky tomu uživatel okamžitě vidí, která konkrétní postava odpovídá, jakému jménu a jaké má atributy.

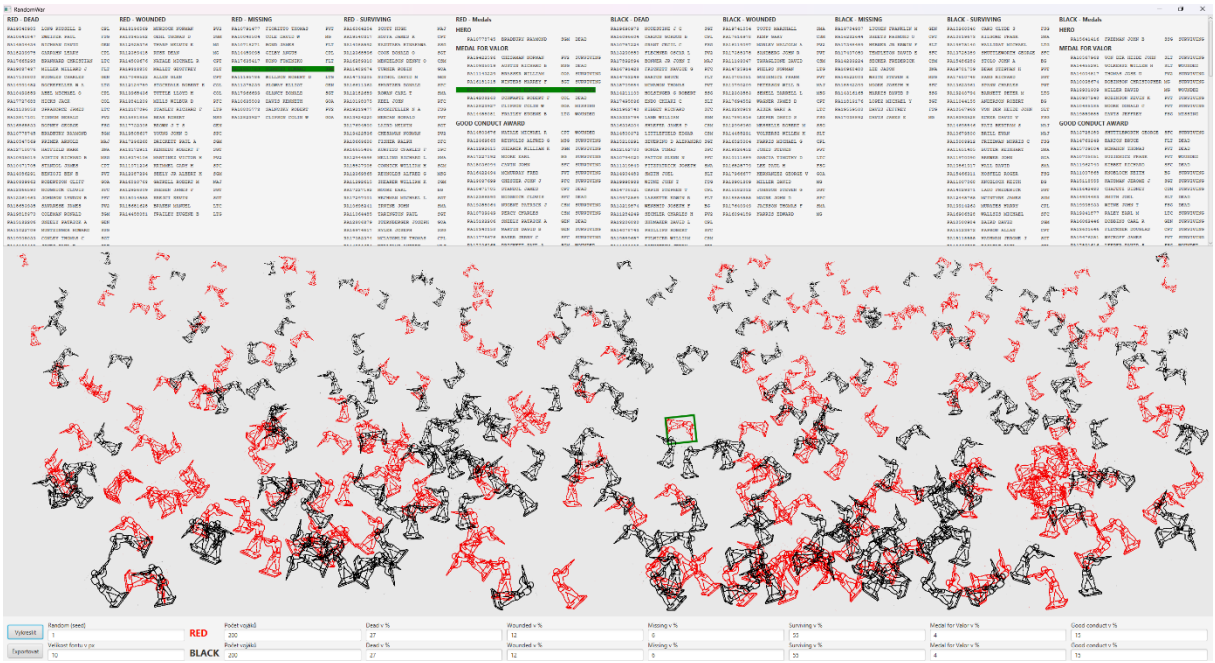
Kód pro zvýraznění vypadá takto:

```
private void highlightRect(Rectangle r, String key, boolean enter) {
    if (enter) {
        r.setStroke(Color.GREEN); // zelené orámování
        r.setStrokeWidth(4);

        var labs = soldierLabels.get(key);
        if (labs!=null) labs.forEach(l-> l.setStyle("-fx-background-color:
green;"));
    } else {
        r.setStroke(null); // zrušení orámování

        var labs = soldierLabels.get(key);
        if (labs!=null) labs.forEach(l-> l.setStyle(""));
    }
}
```

Tato interaktivita je obzvlášť důležitá v kontextu díla *Random War*, které záměrně pracuje s velkým počtem anonymních prvků – vojáků, kteří by jinak mohli působit jako bezejmenná masa. Právě možnost identifikace a zpětného dohledání identity každé jednotky zdůrazňuje individualitu v rámci generovaného davu a odkazuje na tematický rozměr původního díla, v němž každá postava nese své „údaje“, byť byly vytvořeny náhodně.

Kromě této základní interakce uživatel rovněž může měnit vzhled textové části pomocí posuvníku pro nastavení velikosti písma. Tím je zajištěna čitelnost i na různě velkých monitorech nebo při prezentaci. V kombinaci s validací vstupů, která uživatele chrání před zadáním chybných nebo nesmyslných dat, tvoří interaktivita logicky propojený celek, který umožňuje nejen estetické vnímání obrazu, ale také analytickou práci s jednotlivými datovými entitami.

Výsledkem je vizualizační prostředí, které se dynamicky přizpůsobuje uživateli, poskytuje mu okamžitou zpětnou vazbu a propojuje výtvarnou stránku s datovou strukturou – a tím naplňuje podstatu generativního díla jako živého, vícevrstvého systému.



Obrázek 29 – Aplikace Random War při zvýraznění vojáka

Exportování obrazu ve vysoké kvalitě

V aplikaci *Random War* je export obrazového výstupu navržen jako komplexní a vysoce přizpůsobitelný proces, který kombinuje jak vizualizaci postav na výkresové ploše, tak detailní textovou statistiku. Uživatel může jednoduše kliknout na tlačítko **„Export“**, čímž spustí generování výstupního obrázku ve formátu PNG. Tento obrázek je konstruován tak, aby odpovídal grafickému rozdělení známému z originálního díla *Random War* – tedy textové části v horní části a obrazové části s vojáky ve spodní.

Při exportu se nejprve vygeneruje snapshot výkresového panelu, kde jsou vykresleni vojáci včetně všech jejich vizuálních transformací (rotace, velikost, zrcadlení). Aby bylo dosaženo dostatečného rozlišení (typicky přes 13 000 pixelů na šířku), je snapshot dynamicky přepočítáván a škálován podle šířky panelu. Tím je zajištěno, že i velmi rozsáhlé simulace mohou být vytištěny ve velkém formátu bez ztráty kvality.

Nad touto obrazovou vrstvou se následně vykreslují textové sekce – sloupce, které obsahují jmenné seznamy vojáků rozdělené podle statusu a medailí. Každá sekce má vlastní nadpis (např. *RED – DEAD*, *BLACK – SURVIVING*, *RED – MEDALS AWARDED*), pod kterým jsou řazeni jednotliví vojáci v jednotném typografickém stylu. Pokud je seznam příliš dlouhý, automaticky se rozdělí do více sloupců, přičemž nadpis se zobrazí vždy jen v prvním z nich. Speciální pravidla se aplikují pro sekce s medailemi, kde jsou podkategorie (*HERO*, *VALOR*, *GOOD CONDUCT*) typograficky zvýrazněny tučně.

Celý výstup je tedy sestaven jako vertikální spojení textu a obrazu: horní část obsahuje přehledné, strukturované a čitelné datové informace, spodní část pak zachycuje vizuální stránku simulace – prostorově rozmístěné, náhodně orientované a velikostně odstupňované vojáky. Výsledný obrázek je exportován jako bitmapa ve vysokém rozlišení, přičemž jeho vzhled i obsah odpovídají jak estetickým principům generativního umění, tak strukturální přesnosti originálu.

Exportní proces je navržen tak, aby byl dostupný z jakékoli fáze aplikace a nevyžadoval dodatečné vstupy. Zároveň je doplněn o zpětnou vazbu – uživatel je po dokončení operace informován o úspěchu nebo případné chybě. Díky této funkci může uživatel snadno dokumentovat výsledky své simulace, sdílet je, tisknout nebo archivovat, čímž se export stává důležitým nástrojem pro další využití aplikace v oblasti umění, výuky i prezentace.

```
// Upravená metoda btnExportOnAction pro dynamické dělení dlouhých sekcí do více sloupců
public void btnExportOnAction() {
    // === Výběr cílového souboru ===
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Uložit export");
    fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("PNG obrázky", "*.png"));
    fileChooser.setInitialFileName("RandomWar_export.png");
    File file =
fileChooser.showSaveDialog(btnExport.getScene().getWindow());
    if (file == null) return;

    // === 1. Získání snapshotu panelu s vojáky ===
    double paneWidth = pane.getWidth();
    double paneHeight = pane.getHeight();
    double soldierSnapshotScale = 6.0;
    while (paneWidth * soldierSnapshotScale < 13000) {
        soldierSnapshotScale++;
    }
    double exportWidth = paneWidth * soldierSnapshotScale;

    SnapshotParameters snapshotParams = new SnapshotParameters();
    snapshotParams.setTransform(Transform.scale(soldierSnapshotScale,
soldierSnapshotScale));
    snapshotParams.setFill(Color.TRANSPARENT);

    WritableImage soldierImage = new WritableImage((int)(paneWidth *
soldierSnapshotScale), (int)(paneHeight * soldierSnapshotScale));
    pane.snapshot(snapshotParams, soldierImage);
    BufferedImage soldierBuffered = SwingFXUtils.fromFXImage(soldierImage,
null);

    // === 2. Konfigurace vykreslování textu ===
    int fontSize = 30;
    int lineHeight = (int)(fontSize * 1.3);
    int defaultColumnWidth = 790;
    int wideColumnWidth = 1070;
    int xStart = 40;
    int yStart = 40;

    // === 3. Definice sekce (blok textu) ===
    class Section {
        final String title;
```

```
        final List<String> lines = new ArrayList<>();
        final boolean allowMultiColumn;
        Section(String title, boolean allowMultiColumn) {
            this.title = title;
            this.allowMultiColumn = allowMultiColumn;
        }
    }

    List<Section> sections = new ArrayList<>();
    List<Honor> preferredOrder = List.of(Honor.HERO, Honor.VALOR,
Honor.GOOD_CONDUCT);

    // === 4. Generování textových sekcí pro statusy a medaile ===
    for (Side side : Side.values()) {
        for (Status status : Status.values()) {
            List<Soldier> group = soldiers.stream()
                    .filter(s -> s.getSide() == side && s.getStatus() ==
status)
                    .toList();
            if (!group.isEmpty()) {
                int maxLinesPerColumn = (int) ((soldierBuffered.getHeight()
/ 2.0) / lineHeight);
                boolean isLong = group.size() > maxLinesPerColumn;
                Section sec = new Section(side + " - " + status, isLong);
                for (Soldier s : group) {
                    sec.lines.add(String.format("%s  %-20s %-3s",
s.getPersonalNumber(), s.getName(), s.getRank()));
                }
                sections.add(sec);
            }
        }

        Map<Honor, List<Soldier>> medalsGrouped = soldiers.stream()
                .filter(s -> s.getSide() == side && s.getHonor() !=
Honor.NONE)
                .collect(Collectors.groupingBy(Soldier::getHonor));

        if (!medalsGrouped.isEmpty()) {
            Section sec = new Section(side + " - MEDALS AWARDED", true);
            for (Honor honor : preferredOrder) {
                List<Soldier> group = medalsGrouped.get(honor);
                if (group != null && !group.isEmpty()) {
                    sec.lines.add(honor.toString());
                    for (Soldier s : group) {
                        sec.lines.add(String.format("    %s  %-20s %-3s
%s", s.getPersonalNumber(), s.getName(), s.getRank(), s.getStatus()));
                    }
                    sec.lines.add("");
                }
            }
            sections.add(sec);
        }
    }

    // === 5. Odhad výšky pro nejvyšší sloupec ===
    int maxHeightEstimate = 0;
    for (Section sec : sections) {
        int lines = sec.lines.size();
        int maxLinesPerColumn = 50;
        int colCount = (int) Math.ceil((double) lines / maxLinesPerColumn);
        int rowsPerCol = (int) Math.ceil((double) lines / colCount);
```

```
            maxHeightEstimate = Math.max(maxHeightEstimate, rowsPerCol);
        }
        int maxColumnHeight = maxHeightEstimate;
        int finalTextHeight = yStart + (maxColumnHeight + 1) * lineHeight;
        int finalImageHeight = finalTextHeight + soldierBuffered.getHeight();

        // === 6. Vytvoření finálního obrázku ===
        BufferedImage finalImage = new BufferedImage((int) exportWidth,
finalImageHeight, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = finalImage.createGraphics();
        g.setColor(java.awt.Color.WHITE);
        g.fillRect(0, 0, finalImage.getWidth(), finalImage.getHeight());
        g.setColor(java.awt.Color.BLACK);

        // === 7. Vykreslování všech sekcí ===
        int x = xStart;
        for (Section sec : sections) {
            List<String> lines = sec.lines;

            int maxHeight = finalTextHeight - yStart - lineHeight;
            int linesPerCol = Math.max(1, maxHeight / lineHeight);
            int colCount = (int) Math.ceil((double) lines.size() /
linesPerCol);

            for (int c = 0; c < colCount; c++) {
                int y = (c == 0) ? yStart : yStart + lineHeight;
                int from = c * linesPerCol;
                int to = Math.min(from + linesPerCol, lines.size());

                if (c == 0) {
                    g.setFont(new java.awt.Font("SansSerif",
java.awt.Font.BOLD, fontSize));
                    g.drawString(sec.title, x, y);
                    y += lineHeight;
                }

                g.setFont(new java.awt.Font("Monospaced", java.awt.Font.PLAIN,
fontSize));
                for (int i = from; i < to; i++) {
                    String line = lines.get(i);
                    if (line.trim().isEmpty()) {
                        y += lineHeight;
                    } else if (preferredOrder.stream().anyMatch(h ->
h.toString().equals(line.trim()))) {
                        g.setFont(new java.awt.Font("SansSerif",
java.awt.Font.BOLD, fontSize));
                        g.drawString(line, x, y);
                        y += lineHeight;
                        g.setFont(new java.awt.Font("Monospaced",
java.awt.Font.PLAIN, fontSize));
                    } else {
                        g.drawString(line, x, y);
                        y += lineHeight;
                    }
                }

                // Zvětšená šířka pouze pro sekci s medailemi
                boolean isMedalsSection = sec.title.contains("MEDALS AWARDED");
                int colWidth = isMedalsSection ? wideColumnWidth :
defaultColumnWidth;
                x += colWidth;
```

```
            }
        }

        // === 8. Vykreslení snapshotu vojáků ===
        g.drawImage(soldierBuffered, 0, finalTextHeight, null);
        g.dispose();

        // === 9. Uložení do PNG souboru ===
        try {
            ImageIO.write(finalImage, "png", file);
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Export dokončen");
            alert.setHeaderText(null);
            alert.setContentText("Export úspěšně dokončen: \n" +
file.getAbsolutePath());
            alert.showAndWait();
        } catch (IOException e) {
            e.printStackTrace();
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Chyba při exportu");
            alert.setHeaderText("Nepodařilo se uložit obrázek");
            alert.setContentText(e.getMessage());
            alert.showAndWait();
        }
    }
```

Výstup a srovnání s originálem

Výstup aplikace *Random War* ve své současné podobě velmi přesně navazuje na vizuální i konceptuální principy původního díla Charlese Csuriho a Jamese Shaffera z roku 1967. Při srovnání moderního náhledu aplikace (obrázek č. 30) a moderního výstupu (obrázek č. 31) s historickým originálem (obrázek č. 32) lze rozpoznat shodné základní členění kompozice, její strukturu i výtvarné kvality, a zároveň identifikovat některé klíčové rozdíly, které vyplývají z technologického a tvůrčího vývoje.

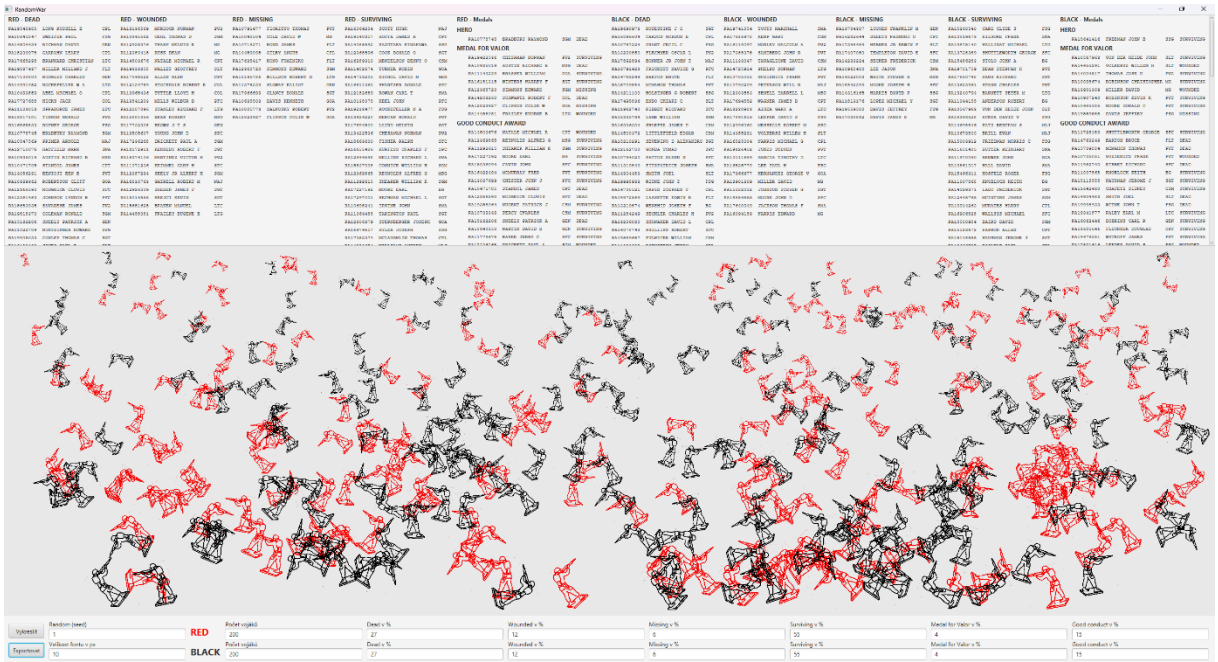
Obě verze jsou rozděleny horizontálně na dvě zcela odlišné vrstvy: horní část obsahuje typografické výpisy dat, zatímco dolní vizualizuje samotné „bojiště“. Horní oblast zobrazuje seznamy vojáků členěné podle statusu (*Dead*, *Wounded*, *Missing*, *Surviving*) a vyznamenání (*Hero*, *Medal for Valor*, *Good Conduct*). V obou případech je datová složka formátována ve sloupcích a působí jako výstup vojenské evidence, doplněná o osobní čísla, jména a hodnosti. Z hlediska kompozice, struktury i funkce je tato část v moderní aplikaci téměř totožná s originálem. Výhodou digitální verze je však větší flexibilita – velikost písma lze měnit a záznamy jsou přímo propojené s obrazem skrze interaktivní zvýraznění.

V dolní části dominuje rozsáhlé pole vojáků, kteří jsou rozděleni barevně na dvě strany – červenou a černou. Stylizace vojáků odpovídá původnímu tvarosloví: jedná se o figurální tvary vytvořené pomocí jednoduchých vektorových tahů. V obou verzích se postavy překrývají, částečně prolínají a tvoří shluky i izolované postavy. Zásadní vizuální efekt v obou případech vytváří perspektivní škálování – postavy směrem k hornímu okraji jsou menší, což vytváří dojem hloubky a prostorového zúžení. Aplikace tento efekt simuluje výpočtem velikosti na základě vertikální pozice, zatímco originál pravděpodobně pracoval s podobnou logikou prostřednictvím plotteru.

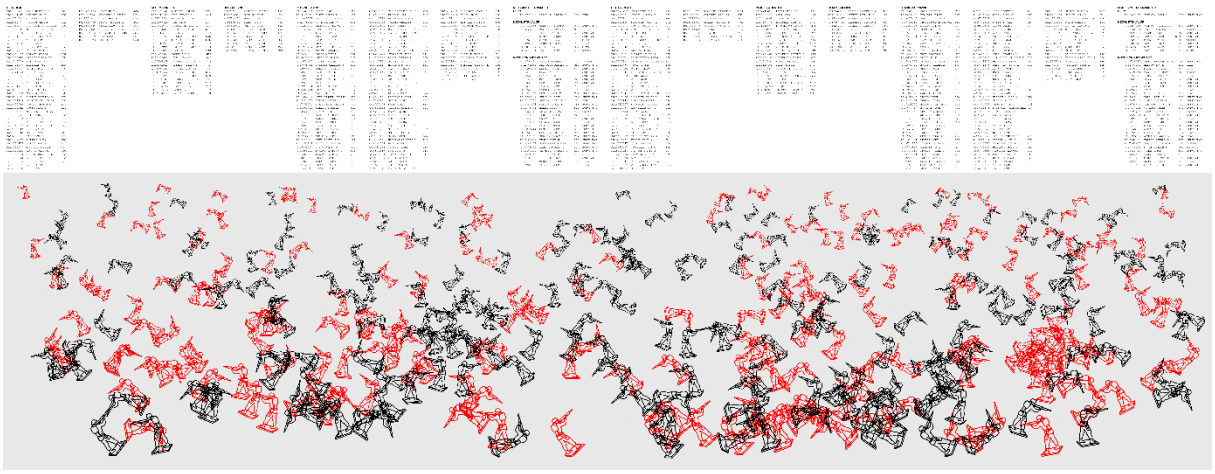
Moderní výstup však působí hustěji a dynamičtěji. Důvodem je vyšší počet vojáků – zatímco originál jich zobrazuje několik stovek, aplikace jich dokáže generovat stovky až tisíce, čímž vzniká bohatší vizuální struktura. V důsledku toho jsou některé oblasti výrazněji zaplněné a dochází k intenzivnějšímu překrývání. Výhodou této hustoty je expresivnější výtvarný dojem, nevýhodou pak možná menší čitelnost jednotlivých postav. Tento rozdíl je však do jisté míry vyvážen interaktivitou – každý voják může být zvýrazněn a identifikován přímo v kontextu výpisu.

Barevné rozlišení jednotek je v obou verzích zachováno – červená proti černé – a jejich rozmístění působí náhodně, ale s viditelnou strukturou. Moderní aplikace navíc přidává jemné variace natočení a horizontálního zrcadlení, díky čemuž působí výstup méně mechanicky a více organicky. Tím vzniká dojem realistického, živého pole, které přesně odpovídá záměru původního díla: propojit náhodné rozdělení s estetikou organizovaného chaosu.

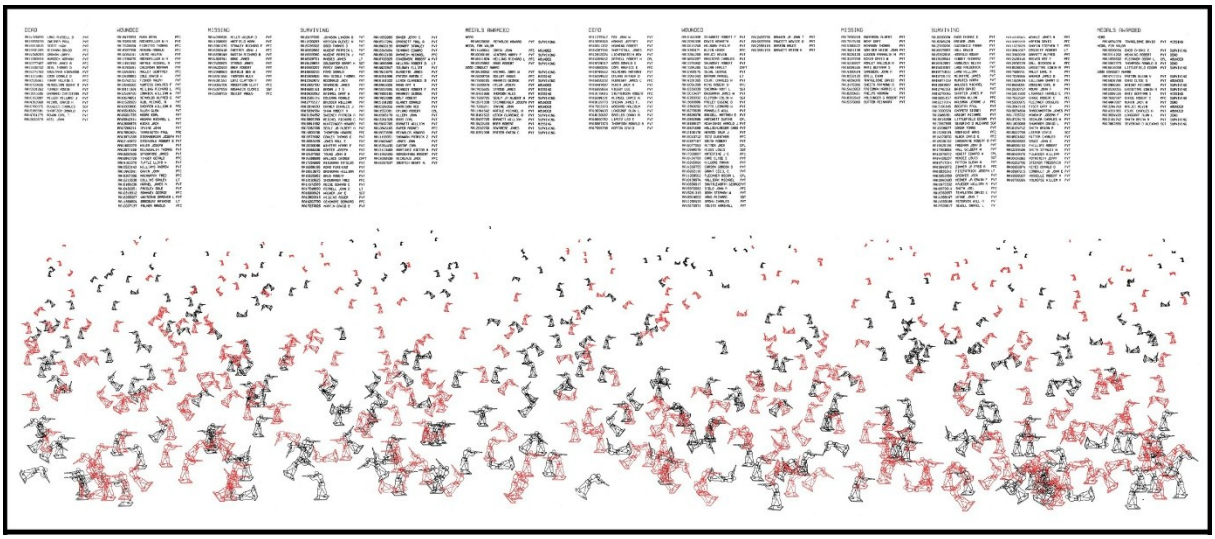
Celkově lze říci, že aplikace věrně zachovává strukturu i význam originálu, ale obohacuje jej o možnosti současných technologií: větší výpočetní kapacitu, flexibilitu rozhraní a přímou interaktivitu. Vizuálně, datově i ideově tak moderní výstup působí jako přesná digitální reinterpretace *Random War*, která si zachovává ducha původního generativního konceptu a zároveň jej rozvíjí do podoby vhodné pro současné vizuální a vzdělávací prostředí.



Obrázek 30 – Random War (náhled vlastní aplikací)



Obrázek 31 – Random War (exportováno z vlastní aplikace)



Obrázek 32 – Random War [23]

2.3 Zhodnocení výsledků

Praktická část práce prokázala, že původní generativní principy Charlese Csuriho lze nejen věrně rekonstruovat, ale zároveň rozšířit a obohatit o nové možnosti díky současným technologiím. Výsledkem je funkční a plně interaktivní aplikace, která vizuálně, datově i konceptuálně naplňuje podstatu původních děl *Flies* a *Random War*, a zároveň nabízí širší škálu uživatelských možností, parametrizaci a okamžitou zpětnou vazbu.

Z hlediska uživatelského rozhraní byla navržena a implementována intuitivní grafická struktura, která umožňuje zadání všech potřebných parametrů – od počtu entit, přes poměrné rozdělení stavů a ocenění, až po vzhled a ovládání vizualizace. Vstupy jsou ošetřeny důslednou validací, čímž je zajištěna stabilita aplikace i při chybném zadání. Ve vizualizační části se podařilo docílit velmi přesvědčivého obrazového výstupu, který kombinuje pseudonáhodné generování s přesným výpočetním modelem – výsledkem jsou esteticky silné, a přitom datově smysluplné kompozice.

Interaktivita aplikace, zejména možnost propojení textového a vizuálního výstupu, výrazně přispívá ke srozumitelnosti a čitelnosti generovaných dat. Každý prvek má přiřazenou identitu a lze jej sledovat jak ve výpisu, tak na obrazové scéně. Uživatel není pouze pasivním pozorovatelem, ale aktivně se podílí na konstrukci výsledku. To posouvá aplikaci nejen do roviny vizuální rekonstrukce, ale také do oblasti didaktické – jako prostředek pro demonstraci nelineárních transformací, pravděpodobnostních rozdělení či algoritmické estetiky.

Z vizuálního hlediska se aplikace velmi dobře přiblížila stylu originálů – zejména v případě *Flies* je podoba téměř identická, s přidanou hodnotou vyššího rozlišení, plynulé interaktivity a přizpůsobitelnosti. V případě *Random War* se podařilo zachovat strukturální i konceptuální vrstvy díla, přičemž aplikace navíc umožňuje generovat mnohem větší množství postav a rozšířit obrazovou složku o nové vizuální variace.

Celkově lze výsledky praktické části hodnotit jako velmi úspěšné. Aplikace splnila všechny stanovené cíle – byla implementována jako simulační nástroj, který dokáže přesně a konzistentně replikovat principy historických generativních děl, a zároveň byla navržena tak, aby s ní mohl pracovat jak odborný uživatel, tak zájemce o digitální umění či datovou vizualizaci. Výsledky potvrzují, že principy generativního umění jsou nejen nadčasové, ale i plně aplikovatelné v dnešním kontextu vizuálního programování, a mohou sloužit jako výchozí bod pro další experimenty, výuku nebo vlastní tvůrčí projekty.

ZÁVĚR

Tato bakalářská práce si kladla za cíl nejen analyzovat tvorbu Charlese Chucka Csuriho jako jednoho z průkopníků generativního umění, ale především navrhnout a realizovat interaktivní simulační aplikaci, která by prostřednictvím současných technologií dokázala nově interpretovat jeho klíčová díla. Kombinací teoretického zázemí a praktické realizace vznikl nástroj, který umožňuje nejen vizuálně reprodukovat principy generativní tvorby, ale také s nimi experimentovat, ovlivňovat je a chápat jejich strukturu zevnitř.

V teoretické části byla nejprve představena historie počítačového umění a klíčová role, kterou v ní Charles Csuri sehrál. Pozornost byla věnována především tomu, jakým způsobem Csuri využíval náhodnost, pravidla a geometrické transformace k vytváření obrazů, které byly esteticky silné, a přitom výpočetně přísně řízené. Rozbor děl *Flies*, *Flies Transformed* a *Random War* položil základy pro návrh aplikace, ve aplikace, ve principy rezonují. Byly popsány matematické modely a datové struktury, které jsou inspirovány těmito obrazy, a popsána metodologie pro jejich převod do moderního programovacího prostředí.

V praktické části byla vyvinuta aplikace v jazyce Java s využitím knihoven JavaFX a FXML. Ta umožňuje generovat obrazy inspirované uvedenými díly na základě náhodných parametrů a uživatelského vstupu. Důraz byl kladen na to, aby výsledné výstupy nejen vizuálně odpovídaly originálům, ale aby zároveň zachovaly logiku a filozofii generativního přístupu: opakovatelnost, variabilitu, parametrické řízení a možnost selektivní transformace. V aplikaci je implementován pokročilý systém validace vstupních hodnot, propracovaný generátor pseudonáhodných čísel, modelování statusů a medailí podle uživatelsky zadaných poměrů, perspektivní škálování, a především – interaktivita, která propojuje grafické výstupy se seznamem dat a umožňuje sledovat identitu každého prvku napříč strukturami.

Výsledky ukázaly, že cíle práce byly splněny. Aplikace dokáže generovat obrazy, které jsou vizuálně i strukturálně velmi blízké originálům, a přitom je rozšiřuje o nové možnosti. Díky interaktivnímu rozhraní, flexibilnímu nastavení parametrů a propojení obrazové a datové vrstvy nabízí hlubší vhled do principů generativní tvorby. Zároveň potvrzuje, že i relativně jednoduché matematické modely mohou vést ke komplexním a výrazově bohatým výstupům.

Osobně vnímám tuto práci nejen jako technický úkol, ale i jako dialog s výtvarnou tradicí digitální estetiky. Během vývoje aplikace jsem měl možnost hlouběji pochopit, jak zásadní roli hraje u Csuriho kombinace řádu a náhody, a jak promyšlené byly jeho algoritmy z hlediska vizuální rovnováhy a kompozice. Realizace aplikace pro mě byla výzvou jak po technické stránce, tak z hlediska estetické citlivosti – bylo třeba nejen programovat, ale i „vnímat obraz“. Práce mi zároveň umožnila propojit mé technické dovednosti s hlubším zájmem o vizuální umění a algoritmické myšlení.

Do budoucna vidím několik možných směrů rozšíření. Prvním přirozeným krokem by bylo rozšíření aplikace o další Csurimu inspirovaná díla, například *Sine Curve Man* nebo *From square to circle (Vitruvian)* a jejich algoritmickou rekonstrukci. Do budoucna by aplikace mohla nabídnout uživateli i možnost navrhovat vlastní transformační rovnice nebo exportovat parametry ve formátu, který by umožnil sdílení a archivaci jednotlivých vizualizací.

Závěrem mohu říct, že práce splnila nejen akademické zadání, ale přinesla mi i osobní uspokojení a nový pohled na propojení výpočetního a vizuálního myšlení. Generativní umění se ukázalo být oblastí, kde se přesně protíná logika s intuicí, a kde může být každý výstup zároveň výsledkem algoritmu i vizuálním zážitkem. Jsem přesvědčen, že aplikace, kterou jsem navrhl, má potenciál zaujmout nejen odborníky, ale i širší publikum se zájmem o vizuální kulturu, výpočetní estetiku a umění v digitálním věku.

POUŽITÁ LITERATURA

[1] SÝKORA, Zdeněk a BLAŽEK, Jaroslav. Computer-Aided Multi-Element Geometrical Abstract Paintings. *Leonardo*. 1970, vol. 3, no. 4, s. 409-413. Dostupné také z: https://doi.org/10.2307/1572257.

[2] SÝKOROVÁ, Lenka. *Zdeněk Sýkora: Životopis*. Online. C2025. Dostupné z: https://www.zdeneksykora.cz/?s=zivotopis. [cit. 2025-05-02].

[3] HENCL, Martin a MAREK, Jaroslav Macrostructures of Zdenek Sýkora. In *19th Conference on Applied Mathematics, APLIMAT 2020 Proceedings*. Bratislava: Slovenská technická univezita v Bratislave, 2020. s. 605–612 s. ISBN 978-80-227-4983-1.

[4] SÝKORA, Zdeněk a KAPPEL, Pavel. *Zdeněk Sýkora 90*. [Praha]: Verzone, 2010. ISBN 978-80-904546-0-6.

[5] MAREK, Jaroslav a NEDVĚDOVÁ, Marie. Pioneering Works in the Application of Random Numbers to Digital Art and Linear Programs of Zdenek Sykora. In *Proceedings of the 32nd Spring Conference on Computer Graphics*. New York: ACM (Association for Computing Machinery), 2016. s. 61-66 s. ISBN 978-1-4503-4436-4.

[6] MAREK, Jaroslav a NEDVĚDOVÁ, Marie. Historie digitálního umění: Náhoda, počítač a Linie Zdeňka Sýkory. In *37. mezinárodní konference Historie matematiky*. Praha: Matfyzpress, 2016. s. 137-146 s. ISBN 978-80-7378-317-4.

[7] SVOBODA, Martin a MAREK, Jaroslav. Construction and statistical analysis of Zdeněk Sýkora's lines. In *APLIMAT 2014: 13th Conference on Applied Mathematics: proceedings*. Bratislava: Slovenská technická univezita v Bratislave, 2014. s. 387-392 s. ISBN 978-80-227-4140-8.

[8] ČTK, KULTURA. *Zemřel průkopník videoartu Woody Vašulka, slavný emigrant od 60. let žil v USA*. Online. 2019. Dostupné z: https://magazin.aktualne.cz/kultura/umeni/woody-vasulka-bohuslav-nekrolog/r~cc61230a24c111ea82ef0cc47ab5f122/. [cit. 2025-05-02].

[9] COMPART – CENTER OF EXCELLENCE DIGITAL ART. *Ben F. Laposky*. Online. [2016]. Dostupné z: http://dada.compart-bremen.de/item/agent/253. [cit. 2025-05-01].

[10] NOLL, A. M. *Computer Art*. Online. ©2019, aktualizováno 8 října 2019. Dostupné z: http://noll.uscannenberg.org/. [cit. 2025-05-02].

[11] RIGHT CLICK SAVE. *An Interview with Frieder Nake*. Online. 2022. Dostupné z: https://www.rightclicksave.com/article/an-interview-with-frieder-nake. [cit. 2025-05-01].

[12] COMPART – CENTER OF EXCELLENCE DIGITAL ART. *Georg Nees*. Online. [2019]. Dostupné z: http://dada.compart-bremen.de/item/agent/15. [cit. 2025-05-02].

[13] THADDAEUS ROPAC. *Vera Molnár*. Online. ©2025. Dostupné z: https://ropac.net/artists/231-vera-molnar/. [cit. 2025-05-01].

[14] BITFORMS GALLERY. *Manfred Mohr: A Formal Language: Celebrating 50 Years of Artwork and Algorithms*. Online. 2019. Dostupné z: https://www.bitforms.art/exhibition/mohr-2019. [cit. 2025-05-02].

[15] GARCIA, Chris. *Harold Cohen and AARON: A 40-Year Collaboration*. Online. 2016. Dostupné z: https://computerhistory.org/blog/harold-cohen-and-aaron-a-40-year-collaboration/. [cit. 2025-05-02].
[16] BROCK, David C. *In Memoriam: Lillian Schwartz, 1927–2024*. Online. 2024. Dostupné z: https://computerhistory.org/blog/in-memoriam-lillian-schwartz-1927-2024/. [cit. 2025-05-02].
[17] CSURI, Caroline. *Charles Csuri: Flies, 1966*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/flies. [cit. 2025-05-02].
[18] COMPART – CENTER OF EXCELLENCE DIGITAL ART. *Charles "Chuck" Csuri*. Online. 2010. Dostupné z: http://dada.compart-bremen.de/item/agent/17. [cit. 2025-05-02].
[19] CSURI, Caroline. *Charles Csuri: Leonardo da Vinci Series, 1966*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/leonardo-da-vinci-series. [cit. 2025-05-02].
[20] REICHARDT, Jasia. *The Computer in Art*. Littlehampton Book Services Lt, 1971. ISBN 9780289795507.
[21] *HERTLEIN, Grace C. (ed.). Computer Graphics and Art. 1976, vol. 1, no. 2.*
[22] CSURI, Caroline. *Charles Csuri: Aging Process Plotter, 1967*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/reviews-1. [cit. 2025-05-02].
[23] CSURI, Caroline. *Charles Csuri: Random War, 1967*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/historic/random-war. [cit. 2025-05-02].
[24] CSURI, Caroline. *Charles Csuri: Hummingbird, 1967*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/historic/hummingbird. [cit. 2025-05-02].
[25] CSURI, Caroline. *Charles Csuri: Sine Curve Man, 1967*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/historic/sine-curve-man. [cit. 2025-05-02].
[26] CSURI, Caroline. *Charles Csuri: 3 Axis Milling Machine, 1968*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/3-axis-milling-machine. [cit. 2025-05-02].
[27] CSURI, Caroline. *Charles Csuri: Mirare, the Digital Art Sculpture, 2010*. Online. [2025]. Dostupné z: https://www.charlescsuri.com/digital-art-sculpture. [cit. 2025-05-02].
[28] HOVEY, Kendra. Beyond boundaries: At the opening of a retrospective of his work, friends and admirers of digital artist Charles Csuri gathered to celebrate his remarkable career. *Columbus Monthly*. Sep. 2010, vol. 2010, no. 9, s. 51–53. Dostupné také z: https://digital-collections.columbuslibrary.org/digital/collection/cbusmonthly/id/60936.
[29] GEEKSFORGEEKS. *Introduction to Java*. Online. 7 března 2025. Dostupné z: https://www.geeksforgeeks.org/introduction-to-java/. [cit. 2025-05-08].
[30] ORACLE. *Java Documentation*. Online. Dostupné z: https://docs.oracle.com/en/java/. [cit. 2025-05-08].
[31] ORACLE. *JavaFX: Getting Started with JavaFX*. Online. ©2008-2014. Dostupné z: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm. [cit. 2025-05-08].
[32] ORACLE. *JavaFX Scene Builder: User Guide*. Online. ©2012-2014. Dostupné z: https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html. [cit. 2025-05-08].

[33] JETBRAINS. *Getting started*. Online. ©2000–2025. Dostupné z: https://www.jetbrains.com/help/idea/getting-started.html. [cit. 2025-05-08].
[34] CSURI, Charles a SHAFFER, James. Art, computers and mathematics. In: *Proceedings of the December 9–11, 1968, Fall Joint Computer Conference, Part II*. San Francisco, California. Association for Computing Machinery, 1968, s. 1293–1298. ISBN 9781450379007. Dostupné z: https://doi.org/10.1145/1476706.1476759.