Motorola Science Cup

Dokumentacja zadania – Gry Wektorowe

Drużyna AC Pajace

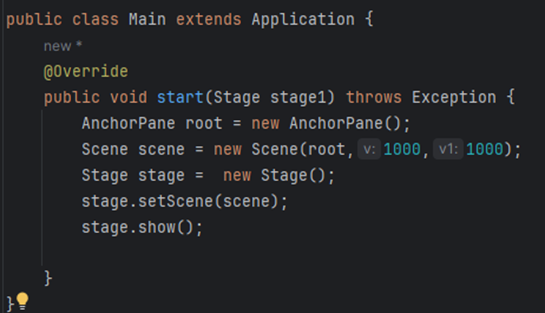
Wstęp

Nasz projekt stworzyliśmy w Javie (w wersji JDK 21), z wykorzystaniem biblioteki JavaFX, ułatwiającej interakcje kodu z obrazem w oknie. Wszystkie grafiki i modele zostały stworzone przez nas w programach InkScape i Blender. Wykorzystane dźwięki zostały stworzone przy użyciu strony sfxr.me lub pobrane ze strony Pixabay.com. Oba te źródła wykorzystują licencję Creative Commons, zezwalającą nam na wolne adaptowanie zasobów w naszym projekcie. W dalszych częściach dokumentacji chcemy opisać działanie i logikę stojącą za działaniem każdej z trzech gier, podzielić się naszymi przemyśleniami i tym czego się nauczyliśmy przez ostatnie kilka miesięcy.

**Uwaga**

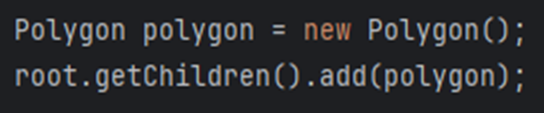
**Aby uruchomić projekt, na komputerze musi być zainstalowana wersja JDK 21! Projekt jest w niej skompilowany!**

Krótko o JavaFX

Nasza aplikacja szeroko wykorzystuje bibliotekę JavaFX i jej klasę Application oraz Timeline. W aplikacji JavaFX klasy AnchorPane, Scene i Stage używane są do zarządzania wyświetlanym obrazem i dodanymi elementami.

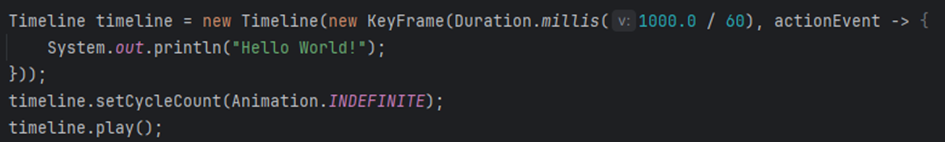
Po uruchomieniu w ten sposób napisanej klasy Main, na ekranie wyświetli się puste okienko o rozmiarze 1000x1000 pikseli.

Aby stworzyć, a następnie dodać widoczne obiekty, np. z klasy Polygon (wielokąty) lub Polyline (odcinki) należy użyć następujących komend:



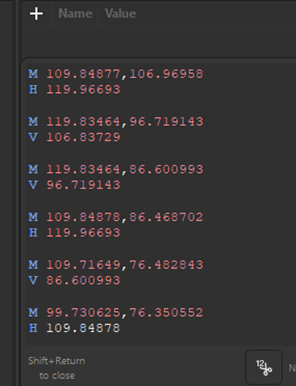
Komenda root.getChildren.remove(polygon); usuwa dodany obiekt z ekranu.

W każdej z gier obiekty wyświetlane na ekranie są instancjami klas extendujących klasę Polygon, albo jako obiekty klasy Polyline (głównie w grze Battlezone). W naszej implementacji ważna też jest klasa Timeline. Symuluje ona klatki, ponieważ pozwala powtarzać wykonywanie kodu co określoną liczbę milisekund.



Uruchomiony w ten sposób Timeline będzie wypisywał ,,Hello World!” do konsoli 60 razy na sekundę, aż nie zostanie zatrzymany komendą timeline.stop();. Obliczenia określające, kiedy i jakie obiekty wyświetlić, wykonujemy własnym kodem, JavaFX jest tylko narzędziem ułatwiającym „rysowanie” obiektów na ekranie.

Inkscape



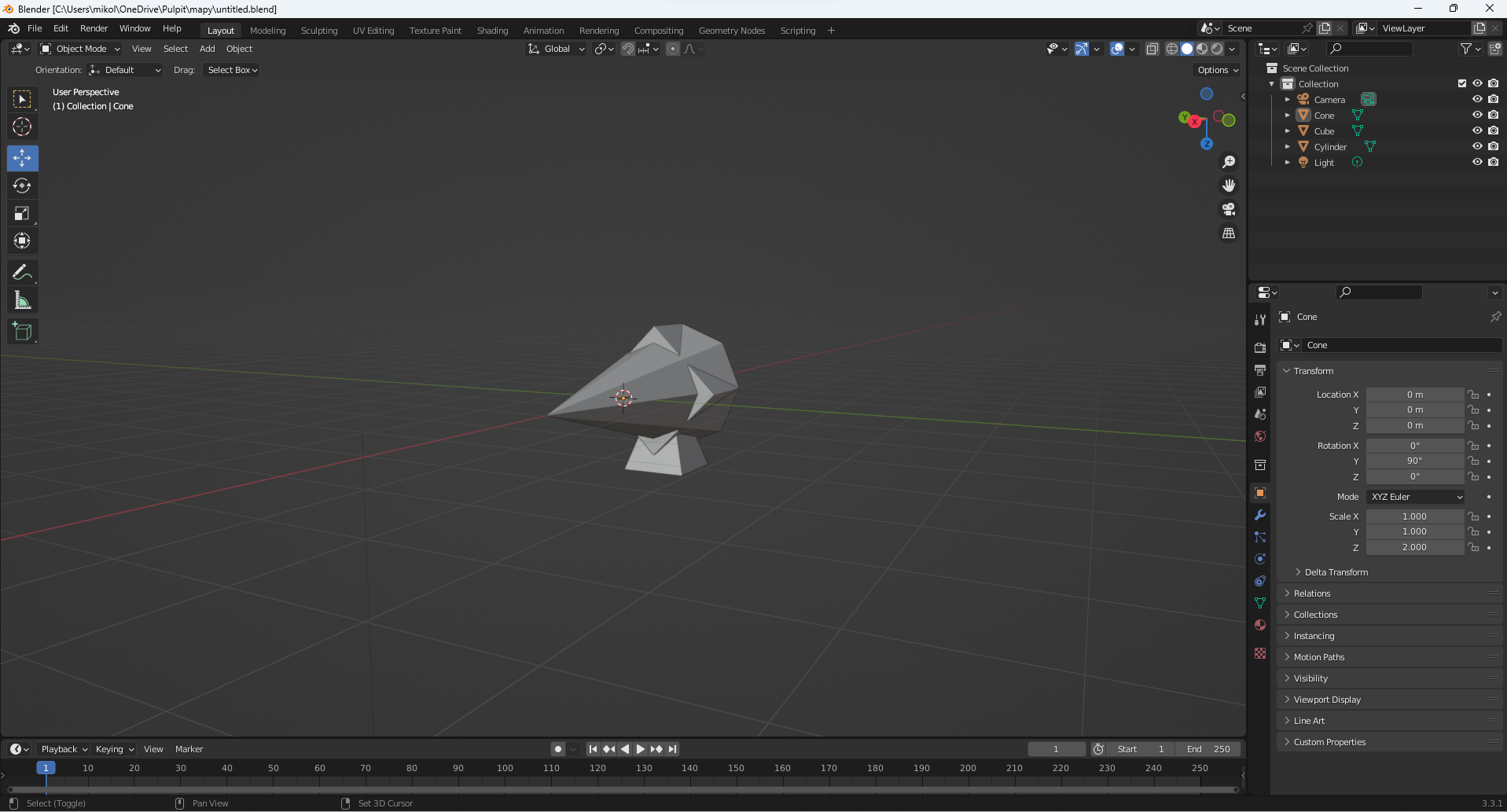
Aby narysować kształty, postacie, horyzont, mapy w grach Tempest i Asteroids użyliśmy aplikacji Inkscape. Umożliwiła nam rysowanie prymitywnych, prostych kształtów wektorowe, chodziło o dokładną lokalizację punktów, które potem nasz kod mógł przeanalizować i narysować. Dokładniej, nasz parser (Util.SVGconverter()) plików .svg z atrybutu ,,path” pliku, otrzymujemy współrzędne punktów, z których stworzony jest rysunek. Następnie możemy wyświetlić na ekranie obiekt złożony z tych punktów jako obiekt klasy Particle, dodając go do roota.

W grze Asteroids większość elementów została narysowana w Inkscape (statki kosmiczne, asteroidy, serca, itd.). W każdej z gier zaczęliśmy od dokładnych oględzin i ograniu żeby poznać, jak najwięcej mechaniki i wyglądu gry. Często polegało to na naszkicowaniu przybliżonej wersji obiektu. Ciężkość polegała na rysowaniu odpowiednich, równoległych kształtów, dlatego zastosowaliśmy metodę rysowania jednej część i odbijania jej względem osi. To pozwoliło nam na równe tworzenie kształtów o tych samych równoległych koordynacjach. W grze Tempest zastosowaliśmy tą samą metodę do stworzenia głównego statku i przeciwników. W BattleZone również użyliśmy Inkscape, animacja pęknięcia po zestrzeleniu gracza.



Blender

Aplikacją, której użyliśmy do tworzenia kształtów w Battlezone to blender. Jako, że gra jest jakby 3D użyliśmy programu, który oferuje nam tworzenie takich obiektów. Znajomość programu Blender umożliwiła nam stworzenie własnych pojazdów. Celem było stworzenie, jak najbardziej odwzorowanych obiektów, które będą miały jak najmniej wierzchołków, by program mógł w łatwy, szybki i ergonomiczny sposób je wygenerować.



Gra 1. – Asteroids

**Sposób stworzenia gry i opis działania kodu**

W naszej implementacji tej gry, każdy występujący na ekranie element (poza tekstem) jest instancją klasy Particle, która jest stworzona na podstawie klasy BetterPolygon, która zaś jest stworzona na podstawie klasy Polygon pochodzącej z JavaFX. Klasa Polygon zawiera listę współrzędnych punktów, z których chcemy utworzyć dany wielokąt. Do tego klasa Particle zawiera wiele ważnych zmiennych:

- listę współrzędnych punktów, z których składa się obiekt (z klasy Polygon),

- obecny wektor prędkości obiektu,

- kąt obrotu,

- przyspieszenie,

- pozycje X i Y

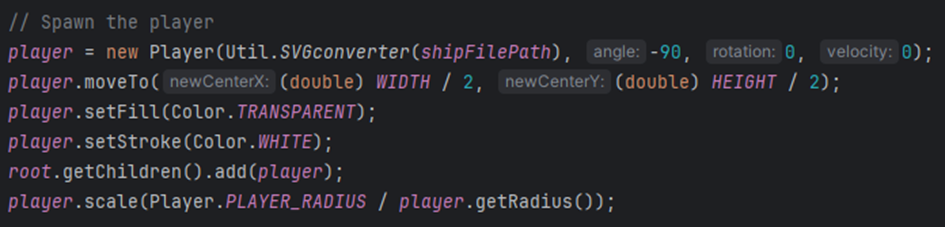
- kolor obiektu

- promień (rozmiar) obiektu

- kilka innych zmiennych, np. poziom tarcia i prędkość rotacji, używane tylko przez statek gracza.

Klasa ta zawiera też metody pozwalające na przesuwanie, skalowanie, obracanie i, co za tym idzie, aktualizację pozycji obiektów co klatkę (na podstawie zmiennych danego obiektu, np. kierunku, prędkości i obecnej pozycji).

Każdy z obiektów (gracz, asteroidy, przeciwnicy, pociski, efekty rozpadu, serca) na początku stworzyliśmy jako rysunek wektorowy w formacie .svg w programie InkScape. Wykorzystując napisany przez nas parser (Util.SVGconverter()) plików .svg z atrybutu ,,path” pliku, otrzymujemy współrzędne punktów, z których stworzony jest rysunek. Następnie możemy wyświetlić na ekranie obiekt złożony z tych punktów jako obiekt klasy Particle, dodając go do roota (klasa Player extenduje klasę Particle, WIDTH i HEIGHT to rozmiary ekranu):



W ten sposób ,,spawnujemy” gracza podczas uruchamiania gry. Każdy obiekt tworzony jest w podobny sposób.

Podczas gry, co klatkę wykrywane są wciśnięcia klawiszy przez użytkownika. Na ich podstawie dochodzi do przyspieszania, obrotu lub strzału pojazdem gracza. Jeżeli gracz nie przyspiesza, zaczyna powoli zwalniać przez tarcie.

← / → = obrót lewo/prawo, ↑ = przyspieszenie statku, X = strzał, E = hyperspace

Funkcja hyperspace teleportuje gracza w losowe miejsce, ale ma pewną szansę zabicia go. Gracz może oddać strzał tylko jeżeli na ekranie jest mniej niż 4 jego pocisków. Pociski znikają po przebyciu określonego dystansu.

Kolizje obiektów wykrywane są wykorzystując funkcję:

Shape.intersects(obiekt1, obiekt2).getLayoutBounds().getWidth()

Funkcja ta sprawdza czy dwa obiekty dodane do roota pokrywają się i zwraca liczbę większą od zera, jeżeli dochodzi do kolizji. Co klatkę sprawdzamy kolizje wszystkich obiektów i na tej podstawie niszczymy i usuwamy obiekty, dodajemy punkty za trafienia lub zabieramy życia w przypadku zderzenia lub trafienia gracza.

Co klatkę aktualizowana jest również pozycja każdego obiektu używając naszej funkcji  
 obiekt.updatePosition();

Jeżeli jakikolwiek obiekt wyjdzie poza pole gry, liczymy na podstawie jego pozycji nową pozycję po drugiej stronie ekranu, do której jest teleportowany. Zachowuje on jednak swój kierunek i prędkość, dzięki czemu wygląda to bardzo naturalnie, jak w oryginalnej grze.

Za pomocą naszych funkcji obiekt.animationParticles() i obiekt.exhaustParticles() tworzymy efekty rozpadania się obiektów po ich zniszczeniu oraz efekt płomienia silnika gracza. Funkcje te tworzą bardzo małe Particles, które mają nadane prędkości i kierunki i znikają po przebyciu określonego dystansu.

**Przebieg i logika gry**

Na początku każdego poziomu spawnujemy gracza i asteroidy, każdą z losową prędkością i kierunkiem poruszania. Gracz musi unikać asteroid i wrogów, aby przeżyć, oraz strzelać do nich, aby zdobywać punkty. Na początku gry otrzymuje 3 życia, po utracie których kończy się gra. Po każdej śmierci (utracie 1 życia), gracz respawnuje się tylko jeżeli w strefie respawnu nie ma przeszkód.

Podczas rozgrywki losowo może pojawić się przeciwnik. Są ich 2 typy: pierwszy, dużo częstszy – strzela losowo i drugi – rzadki, ale strzelający do gracza bardzo celnie. Przeciwnicy strzelają z określoną częstotliwością i skręcają w losowe kierunki. Wraz z postępem gry szansa pojawienia się typu 2 wzrasta. Przeciwnik przelatuje z jednej strony ekranu na drugą, po czym znika. Za zabicie wroga przyznawane są punkty.

Zniszczenie wszystkich asteroid i wrogów, powoduje uruchomienie następnego poziomu. Każdy następny poziom ma większą liczbę początkowych asteroid. Co 10000 punktów gracz otrzymuje 1 dodatkowe życie. Po zniszczeniu dużej lub średniej asteroidy ,,rozpada” się ona na 2 mniejsze. Małe asteroidy nie dzielą się dalej. HUD wyświetla obecną liczbę żyć w postaci serc, high-score, obecną liczbę punktów oraz guzik wyjścia z gry.

Po zakończonej rozgrywce, jeżeli zdobyta liczba punktów jest większa niż obecny zapisany high-score, zostaje ona zapisana jako nowy rekord. Pojawia się ekran Game Over, gdzie możemy wybrać ponowną rozgrywkę, albo powrót do menu głównego. Jako nasz dodatek kreatywny, dodaliśmy asteroidę leczącą, która ma szansę pojawić na początku każdego poziomu i jest czerwona, z sercem w środku. Zniszczenie jej dodaje 1 życie.

Gra 2. - Tempest

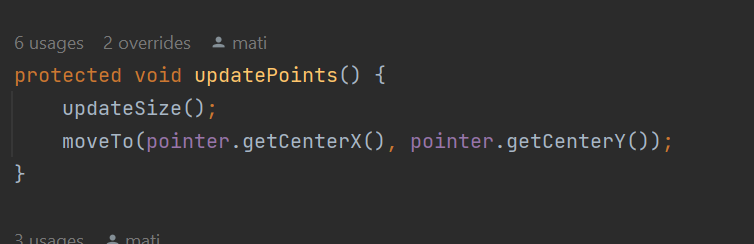
**Sposób stworzenia gry i opis działania kodu**

Obiekty występujące w naszej implementacji gry, są instancjami klasy Enemy, która jest instancją klasy *BetterPolygon* stworzonej na podstawie klasy Polygon wywodzącej się z JavaFX. Wyjątkiem jest klasa Player która powstała na wzór klasy *BetterPolygon*. Oprócz punktów obiektów zawartych w klasie Polygon, Enemy ma również inne zmienne, potrzebne aby obiekty funkcjonowały prawidłowo, na przykład:

* Obecny Panel na którym jest obiekt
* Obecna wysokość obiektu w panelu
* Pointer
* Cooldown pocisków

Pointer służy do obliczania współrzędnych do których przeciwnicy mają się poruszać. Obliczamy położenie pointera za pomocą *metody updatePointer().*





W metodzie update points pointer jest używany do uzyskania koordynatów, do których *Enemy* ma się przenieść w następnej klatce.

Ponadto *Enemy* posiada dwie klasy wewnętrzne *Bullet* oraz *Seed*. Klasa Bullet odpowiada za tworzenie oraz poruszanie się pocisków przeciwników dzięki metodom takim jak spawn() oraz move(). *Seed* reprezentuje punkty tworzone na początku poziomu z których potem powstają przeciwnicy.

Wszystkie użyte przez nas obiekty zostały zaprojektowane w aplikacji InkScape, skąd potem za pomocą naszego parsera (*Util.SVGconverter()*) zostały zamienione w Polygon oraz wyświetlone na ekranie.

Każda mapa w grze została przez nas podzielona na 16 Polylineów - Paneli. Każdy Panel posiada :

* Listy wszystkich wrogów znajdujących się na panelu
* Listę pocisków gracza oraz przeciwników
* Kąt na którym znajduje się panel względem horyzontalnej linii
* Długość
* Kolor

Do tworzenia mapy i paneli stworzyliśmy klasę *Graphics* odpowiedzialną m.in. za tworzenie mapy dzięki metodzie *drawMap()* oraz paneli metodą *createPanels().*

Klawisze ←/→ gracz porusza się w prawo/lewo a za pomocą klawisza “*X*” wykonuje strzał w głąb panelu. Dodatkowo klikając klawisz *“E*” gracz aktywuje *Superzapper* który niszczy wszystkie wrogie obiekty na mapie jednak jest dostepny tylko raz na poziom.

**Przebieg i logika gry**

Na początku każdego poziomuje gracz zostaje zespawnowany na jednym z panelów, a przeciwnicy pojawiają się jako kolorowe punkty (seeds) w miniaturyzacji mapy znajdującej się zazwyczaj niedaleko środka kształtu. Poruszając się w kształt samej mapy każdy przeciwnik trafia w końcu do dolnej części losowego panelu skąd porusza się w górę lub w bok w zależności od rodzaju. W grze występują 4 rodzaje przeciwników.

Flipper - najczęstszy z przeciwników występuje w kształcie czerwonej muszki. Przemieszcza się w górę czasami zmieniając panele po drodze. Po dotarciu na szczyt goni gracza po górnej krawędzi. Strzela również shurikenami w górę panelu na którym się znajduje.

Tanker - fioletowy kształt który porusza się tylko w górę panelu.Gdy tanker dotrze do 80% wysokości panela, zamienia się on na 2 Flipper na sąsiednich panelach. Również strzela w kierunku gracza.

Spiker - wyglądem przypomina zielone jojo. Jako jedyny z przeciwników nie pojawia się jako seed tylko jako zielony punkt na dole losowego panelu. Porusza się w górę zostawiając za sobą zieloną linie (kolce), nie dociera jednak na samą górę tylko po osiągnięciu odpowiedniej wysokości zawraca do początku panelu. Po dotarciu z powrotem na początek segmentu spiker zamienia się w tankera. Podczas zmiany mapy gracz musi omijać lub zniszczyć kolce pozostawione przez spikera ponieważ, kolizja z nimi skutkuje utratą życia. Spiker również strzela w kierunku gracza.

Fuseball - ostatni oraz najrzadszy z przeciwników jest zbiorem kolorowych iskierek. W przeciwieństwie do reszty nie porusza się pośrodku panelu tylko po linii bocznej oddzielającej dwa segmenty od siebie. Poruszając się do góry jest najszybszym z obiektów lecz gdy zmienia panel znacząco zwalnia. Po dotarciu na górę, podobnie jak flipper, goni gracza. Jako jedyny nie strzela w kierunku statku.

Przejście poziomu następuje w dwóch przypadkach, kiedy wszyscy przeciwnicy zostaną zniszczeni lub, gdy wszystkie pozostałe flippery dostaną się na skraj mapy.

z każdym kolejnym poziomem ilość wzrasta oraz zwiększa się szansa na pojawienie się tankerów, spikerów lub fuseballi.

HUD wyświetla liczbę żyć w postaci serduszek w lewym górnym rogu ekranu wyświetla również liczbę zdobytych punktów w aktualnej rozgrywce, high-score, przycisk wyjścia z gry, a także koło informujące o możliwości użycia Superzappera

Po przegranej jeżeli został pobity high-score zostaje on zastąpiony. Pojawia się ekran Game Over, gdzie możemy zagrać ponownie albo wrócić do menu.

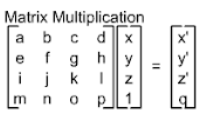
Gra 3. – Battlezone

**Sposób stworzenia gry i opis działania kodu**

Battlezone jest wyjątkowy, ponieważ jest prawdopodobnie pierwszą grą 3D typu First Person Shooter w historii. Stworzenie go przez nas wymagało nauczenia się konceptu i metod projekcji przestrzeni 3D na ekran 2D, z wykorzystaniem mnożenia matryc translacji, rotacji, skali, kamery, widoku i projekcji przez współrzędne punktów w przestrzeni 3D. Było to bardzo ciekawe i pozwoliło nauczyć się nam wiele nowych rzeczy o grafice 3D i nowych dla nas konceptach matematycznych.

Podczas gdy obiekty w 2 poprzednich grach były instancjami klasy BetterPolygon (lub Particle), wszystkie obiekty tej gry należą do klasy Object3D. Klasa Object3D zawiera wszystkie ważne informacje o obiekcie 3D:

* listę współrzędnych XYZ wierzchołków, z których zbudowany jest obiekt
* listę twarzy obiektu - twarz zawiera listę wierzchołków, które należy połączyć
* obecną pozycję i rotację obiektu

Do tego klasa ta zawiera metody pozwalające na przesuwanie, obrót, skalowanie i wyświetlanie obiektu.

Każda z wyżej opisanych operacji wymaga przemnożenia współrzędnych każdego wierzchołka obiektu przez odpowiednią matrycę. Szablon mnożenia jest widoczny na rysunku. Matryce te zawierają odpowiednie zmienne, zdefiniowane matematycznie.

Najciekawszą z tych operacji jest wyświetlanie obiektu. Potrzebny do tego jest obiekt kamery (Camera - obiekt z tej klasy pełni rolę gracza), z którego perspektywy będziemy obserwować wirtualną przestrzeń 3D. Aby wyświetlić na ekranie (który jest płaszczyzną 2D) obiekt z przestrzeni trójwymiarowej należy wymnożyć wierzchołki obiektu po kolei przez te matryce:

* matrycę kamery - pozwala ona zinterpretować stosunkową pozycję obiektu i kamery, która go obserwuje
* matrycę projekcji - dzięki niej otrzymujemy ,,rzut” wierzchołków na przestrzeń 2D ekranu poprzez pobranie współrzędnych X i Y, ze współrzędnych XYZ wierzchołka. Jeżeli wartość bezwzględna którejkolwiek z XYZ jest większa niż 1, należy odrzucić i nie wyświetlać tego wierzchołka, ponieważ ,,nie mieści” się on na ekranie.
* matryca wyświetlacza - dzięki niej punkty otrzymują odpowiednią pozycję XY(lewo/prawo i góra/dół) na ekranie użytkownika, gdzie punkt (0,0) to lewy górny róg, a punkt (Xmax, Ymax) to prawy dolny róg ekranu

Po wykonaniu tych operacji otrzymane punkty należy połączyć liniami na podstawie ich przynależności do listy twarzy danego obiektu. Wtedy na ekranie wyświetli się dany obiekt, widziany z punktu widzenia kamery w przestrzeni 3D.

Nasze obiekty stworzyliśmy w Blenderze i wyeksportowaliśmy w formacie .obj, który zawiera wszystkie potrzebne nam informacje na temat modelu (listę wierzchołków i twarzy). Nasz parser interpretuje pliki i na tej podstawie tworzy obiekty.

Kolizje obiektów są wykrywane w specjalny sposób - ponieważ w grze zarówno gracz, jak i jego pociski poruszają się tylko w 2 płaszczyznach (przód/tył i lewo/prawo) i nie poruszają się góra/dół, obliczenia kolizji wystarczy zatem liczyć tylko w tych właśnie 2 płaszczyznach, a nie w 3. Pozwala to zaoszczędzić ogromne ilości mocy obliczeniowej, ponieważ wykrywanie kolizji 3D jest dużo bardziej kosztowne pod tym względem (w porównaniu do kolizji 2D). Każdy obiekt ma swój ,,hitbox” i za pomocą funkcji obiekt.runCollisionCheck(), może wykryć kolizje z innymi obiektami i odpowiednio zareagować (pociski używają funkcji pocisk.checkForHits()). Detekcja polega na sprawdzaniu przecięć linii składających się na hitboxy, przy użyciu algorytmu wykrywania przecięć odcinków (Util.lineIntersect()).

Generacja świata wykorzystuje system Chunków inspirowany m.in. Minecraftem. System ten ładuje tylko obiekty, które są dostatecznie blisko. Jeżeli gracz oddali się za bardzo od danego chunka, obiekty znajdujące się w nim są ,,despawnowane”. Jednocześnie po pojawieniu się gracza na nowym terenie, pusty Chunk blisko gracza zostaje załadowany wraz z obiektami. System ten pozwala na praktycznie nieskończone pole rozgrywki, bardzo niskim kosztem obliczeniowym.

Sterowanie : W - jazda do przodu, S - jazda do tyłu, Q - obrót w lewo, E - obrót w prawo, A - jazda po ukosie w lewo, D - jazda po ukosie w prawo, S+Q - jazda po ukosie (do tyłu i w prawo), S+E - jazda po ukosie (do tyłu i w lewo), SPACJA - strzał

**Przebieg i logika gry**

Na samym początku gry ładowane są pobliskie chunki oraz gracz. Gracz otrzymuje 3 życia, po których utracie przegrywa. Na polu gry obecne są przeszkody (statyczne elementy otoczenia) oraz miny - najechanie na minę odbiera jedno życie (miny są naszym dodatkiem kreatywnym do gry). Jeżeli na mapie nie ma przeciwników lub UFO (klasa EnemyTank), mogą się one losowo zespawnować (na początku gry, lub po wyeliminowaniu wrogów). Strzał można oddać tylko w określonych odstępach czasu (dochodzi do przeładowania naboju w magazynku). Gracz strzela do wrogów, aby zdobywać punkty i unika ich ataków, aby przeżyć. W grze obecni są 3 agresywni wrogowie (Czołg, SuperCzołg, Rakieta Samonaprowadzająca) i jeden neutralny (UFO). Wraz ze zdobywaniem punktów przeciwników pojawia się więcej i częściej pojawiają się SuperCzołgi i Rakiety. Radar u góry ekranu pokazuje pozycję agresywnych wrogów na mapie (bez min i UFO). HUD wyświetla rekord gracza w tej sesji, obecną liczbę punktów i liczbę pozostałych żyć oraz guzik wyjścia do Menu Głównego. Po utracie 3 żyć gracz może zagrać ponownie albo wrócić do menu.

Dźwięk

Dźwięk w naszej grze tworzony jest przez klasę Sound, przy użyciu wbudowanej klasy Clip w Javie. Stworzone przez nas lub pobrane pliki dźwiękowe przechowujemy w formacie .wav i puszczamy lub stopujemy w odpowiednich momentach w kodzie.

Podsumowanie

Dzięki udziałowi w tym konkursie przez ostatnie kilka miesięcy bardzo dużo się nauczyliśmy. Była to wspaniała okazja do pogłębienia zarówno swojej wiedzy programistycznej i matematycznej, jak i naszych zdolności do pracy zespołowej. Mimo, że momentami wyzwania stojące przed nami wydawały się zbyt trudne, ostatecznie ukończyliśmy nasz pełen projekt.

Screeny ze stworzonych przez nas gier.

