Dokumentacja Projektu Gra Cywilizacja 2077

Konrad Wołoszyn

Jessica Adamczyk

Jan Palmen

Opis Gry

Gra polega na rozbudowie swojej cywilizacji poprzez zakup odpowiednich budynków i rozmieszczanie ich w najkorzystniejszy sposób. Budynki te dzielą się na 4 kategorie. Każda oznacza inny rodzaj dołączonej funkcjonalności:

a) **Bank –** jest odpowiedzialny za zwiększanie maksymalnej wartości pieniędzy, jaką gracz może posiadać.
b) **Kasyno –** zwiększa ilość zarabianych co rundę pieniędzy
c) **Koszary –** są odpowiedzialne za umieszczanie na planszy robotów, czyli jednostek gracza
d) **Laboratorium –** zwiększa zasięg ruchu danego typu robotów

Każdy z budynków posiada punkty wytrzymałości. Oprócz tych budynków gracz posiada również stolicę, która odpowiada za ogólne punkty życia gracza. Po utracie stolicy gracz przegrywa grę. Innymi obiektami stawianymi przez gracza są roboty. Mogą one zostać umieszczone tylko, jeżeli gracz posiada Koszary. Roboty dzielą się na: ofensywne i defensywne.

Wrogimi obiektami są:

a) **Stolica wroga –** jest odpowiedzialna za punkty życia wroga. Po zniszczeniu jej, gracz wygrywa grę
b) **Wrogi budynek –** służy jako przeszkoda, ale jest również odpowiedzialny za ilość generowanych przez wroga robotów
c) **Wrogi robot –** jednostka mogąca się poruszać oraz atakować wszystkie obiekty gracza

Rozgrywka

Gracz umieszcza na planszy budynki i przygotowuje się do ataku na wrogą stolicę, przy okazji starając się zabezpieczyć własną. Atak na wrogie jednostki odbywa się za pomocą posiadanych robotów. Aby przemieścić robota, należy wybrać pole, na które ma się przemieścić i zakończyć swoją turę przyciskiem „Następna Tura”. Jeśli po drodze napotka wrogi budynek lub wrogiego robota, dochodzi do bitwy, w której na przemian jednostki zadają sobie obrażenia, do momentu, aż życie którejś z nich wyniesie 0. Wróg w swojej turze kieruje roboty na jednostki gracza i w przypadku toru kolizyjnego, potyczka działa na tej samej zasadzie. Gra toczy się do momentu, w którym roboty którejś ze stron zniszczą wrogą stolicę.

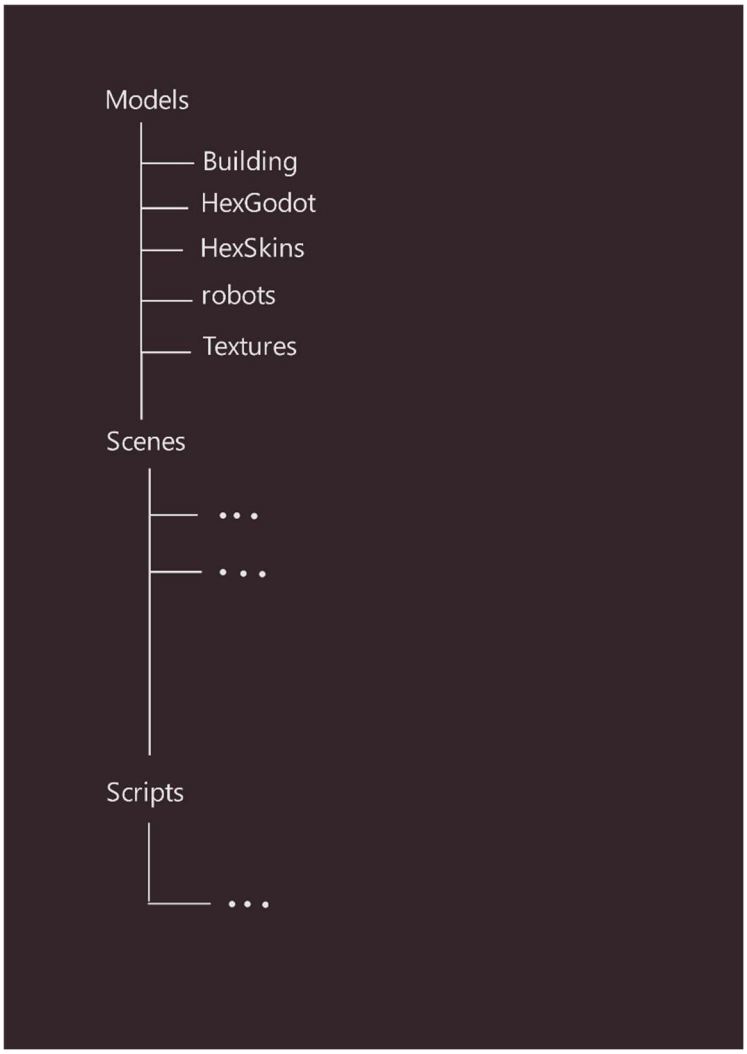
Gry Typu 4x

Gra, którą stworzyliśmy wpisuje się w schemat gier typu 4x. Za nazwą 4x stoją wyrazy: Explore, Expand, Exploit oraz Exterminate, które określają główne cechy tego gatunku. Zamysłem jest

eksploracja świata i środowiska, w którym znajduje się gracz, rozszerzanie swojego terytorium, wykorzystywanie zasobów oraz dążenie do pokonania przeciwnika. Na samym wstępie gracz ma możliwość zapoznania się ze światem, w którym gra się odbywa w celu obrania najlepszej strategii rozwoju kontrolowanej przez siebie cywilizacji. Typ 4x, mimo, że daje znaczne pole do obrania swojej własnej interpretacji, stawia jednak pewne wymogi. Po pierwsze gracz musi mieć możliwość nawigowania i wyświetlania fragmentów większej i zróżnicowanej mapy. Zalecane jest również wprowadzenie rozmaitych środowisk posiadających własne charakterystyczne cechy. Istotne jest danie możliwości nie tylko rozszerzania, ale również ulepszania posiadanego już terytorium na podstawie zdobywanych przez gracza punktów. Należy również wziąć pod uwagę fakt, że gry tego typu odbywają się na zasadzie turowej, co daje możliwość symulacji upływającego czasu.

Struktura projektu

Projekt został podzielony na odpowiednie części ze względu na typ oraz przeznaczenie danych plików i folderów.



1. **Models**

   W folderze Models znajdują się pliki modeli 3D oraz wykorzystywanych tekstur.

   a. W folderze Building przechowywane są modele budynków dostępnych w trakcie gry. Modele te są generowane poprzez kod w odpowiednich momentach.
   b. W folderze HexGodot znajdują się struktury heksagonalne, które posłużyły w celu stworzenia mapy. Odpowiadają one za wygląd planszy, są generowane poprzez kod.
   c. HexSkins jest folderem, w którym znajdują się tekstury wykorzystywane poprzez struktury folderu HexGodot.
   d. Robots zawiera modele poszczególnych robotów.
   e. Textures zawiera tekstury wykorzystywane w projekcie. Są to głównie tekstury interfejsu użytkownika.

2. **Scenes**

   Folder Scenes zawiera definicje scen w Godot. Jest to zarówno scena główna (uruchamiana jako pierwsza po włączeniu programu) jak i sceny przez nią wykorzystywane. Godot pozwala na tworzenie scen i późniejsze ich użycie w innych scenach. Jest to możliwe zarówno bezpośrednio w edytorze, jak i poprzez wygenerowanie poprzez kod. Dzięki opcji generowania poprzez kod możliwe jest generowanie instancji scen, co z kolei ułatwia np. proces umieszczania wielu jednostek danej struktury w sposób dynamiczny.

3. **Scripts**

   W tym folderze znajdują się wszystkie wykorzystane skrypty. Są one odpowiedzialne za różne aspekty gry, takie jak interpretowanie wyborów użytkownika, zmianę wyglądu interfejsu w czasie rzeczywistym, generowanie nowych struktur, obsługiwanie danych na temat umieszczonych obiektów.

Dobór wzorców architektonicznych

W naszym projekcie, ze względu na to, że jest to gra, wykorzystaliśmy wzorzec zbliżony do wzorca ECS (Entity-component-system). Wzorzec ten pozwala na dużą elastyczność podczas rozwijania projektu w poszczególnych etapach. Skupia się on na trzech ważnych elementach: jednostkach, komponentach i systemach. Komponenty cechują się tym, że są one zdefiniowane w najprostszej postaci, tak by mogły być wielokrotnie używane. Nie posiadają one własnej logiki, przez co stają się bardziej wszechstronne. W przypadku naszego projektu do kategorii komponentów można zaliczyć wszystkie struktury znajdujące się w swoich własnych scenach. W każdym momencie mamy możliwość wygenerowania w kodzie danej sceny, jak również możemy umieścić ją jako część innej sceny, bezpośrednio w edytorze. Jednostki składają się z wielu komponentów. W przypadku naszego projektu głównymi jednostkami są menu wyboru akcji oraz mapa. Korzystają one swobodnie z dostępnych scen, przez co nie jest konieczne statyczne umieszczanie tych scen bezpośrednio w poszczególnych edytorach. System został utworzony głównie na zasadzie globalnego skryptu, który nie jest, jak inne skrypty, przypisany do danego obiektu, przez co jest również niezależny od wywołania innych obiektów. Działa on przez cały czas i jest odpowiedzialny za zarządzanie danymi, jak również umożliwienie komunikacji pomiędzy skryptami, które są zależne od wywołania instancji danej sceny. Pozostałe pliki skryptowe są również częścią systemu i są odpowiedzialne za poszczególne funkcjonalności, takie jak np. umieszczanie budynków, generowanie mapy czy aktualizowanie statusu wyświetlanych informacji.

Zastosowane Struktury Danych

W naszym projekcie, ze względu na sposób wywoływania kolejnych instancji danych obiektów oraz konieczność ciągłego monitorowania sytuacji na mapie, zdecydowaliśmy się na wykorzystanie list, jako kontenerów na dane. Elementy list są reprezentacją danych pól na mapie. Dzięki temu mogliśmy w każdym momencie, w razie potrzeby zmieniać informacje na temat obiektu znajdującego się na danym polu mapy. Stało się to bardzo istotne w momencie sprawdzania dostępnych pól, jak również w przypadku interpretowania kolizji walczących ze sobą robotów. Dzięki iterowaniu poprzez odpowiednie listy mogliśmy porównać ścieżki danego robota z pozycjami wrogich robotów i na podstawie pozyskanych informacji rozpatrzyć na podstawie tych samych indeksów w listach siły i punktów-życia potencjalną potyczkę pomiędzy tymi robotami. W celu łatwiejszego poruszania się poprzez listę, zinterpretowaliśmy mapę jako ciąg liczb, a następnie zdefiniowaliśmy funkcję zmieniającą wektor pozycji na mapie na indeks listy.

Funkcje Wykorzystane W Projekcie

Hexagon Map Generator – skrypt ten odpowiada za generowanie z pomocą GridMap losowej heksagonalnej mapy wedle ustalonych założeń: losowania miejsca pola zieleni (gdzie

gracz może budować), następnie niewielkich pól wody (teren niedostępny), a na końcu wypełnia pozostałe pola pustynią (można się po nich przemieszczać, jednak bez możliwości budowania).

Camera Node Movement – skrypt odpowiadający za poruszanie kamerą za pomocą przesuwania, a także za ogólną interakcję z myszą. Reaguje na klikanie gracza, zwracając indeks klikniętego heksagonu oraz na tej podstawie wyzwalając różne funkcje, w zależności od tego co stoi na tym indeksie, od wyświetlania opisu, ilości HP, siły po wyświetlanie menu zakupowego.

Dijkstra – skrypt przechowujący klasę algorytmu Dijkstry poszukującego najszybszej drogi na heksagonalnej mapie.

Global –skrypt przechowujący funkcje używane globalnie, czyli w wielu innych skryptach, które po prostu odwołują się do Globala. Znajdą się tu takie funkcje jak: konwertowanie współrzędnych mapy na indeksy obiektów i odwrotnie, wywoływanie w poprawny sposób algorytmu Dijkstry, anulowanie ruchu robota, pobieranie indeksu obiektu z Mesh Library (aby postawić go po samej nazwie), zmienianie opisu oraz aktualizowanie widoku posiadanych pieniędzy.

Grid Container Building – reaguje na menu pozwalające na budowanie obiektów za opłatą.

Grid Container Bank – reaguje na menu pozwalające na kosztowne ulepszanie pojemności i sprzedaż klikniętego banku.

Grid Container Barracks – reaguje na menu pozwalające na kosztowną rekrutację robotów na wolnym miejscu wokół koszar, jeśli takowe istnieje.

Grid Container Kasyno – reaguje na menu pozwalające na zakup ulepszenia dochodów i sprzedaż klikniętego kasyna.

Grid Container Laboratory – reaguje na menu pozwalające na zakup ulepszenie siły i życia poszczególnych robotów oraz na zwiększenie ogólnego zasięgu ruchu robotów.

Grid Container Robot – pozwala przestawienie robota z pomocą algorytmu Dijkstry.

Grid Container Next Round – wyzwalany przyciskiem następnej rundy, przelicza, ile kasyn i banków ma gracz, a następnie na tej podstawie zwraca graczowi bilans funduszy na następną rundę. Dodatkowo odpowiada za wykonanie wszystkich ruchów robotów gracza i przeciwnika, i sprawdzenie czy na coś nie nachodzą. Jeśli najdą na wrogie jednostki bądź budynki, dochodzi do przeliczenia HP oraz siły, a następnie śmierci słabszego. Jeśli zostanie zniszczona stolica jednego z nich, to następuje koniec rozgrywki i wygrana, bądź przegrana.

Testy interfejsu

Na bieżąco wykonywane zostały testy zachowania interfejsu użytkownika, w głównej mierze menu wyboru akcji. W wyniku napotykanych problemów, podjęte zostały kroki, by uniemożliwić spowodowanie nieprawidłowości działania całego systemu poprzez interakcje użytkownika z interfejsem. Zabezpieczenia powodują, że gracz może tylko wykonywać daną czynność, tylko jeżeli dane warunki zostaną spełnione.

Testy Mechaniki Umieszczania Jednostek

Funkcje służące do umieszczania budynków oraz robotów zarówno przez gracza, jak i komputer zostały przetestowane wielokrotnie, w celu sprawdzenia poprawności umieszczania jednostek na odpowiednich polach oraz w celu sprawdzenia blokady możliwości stawiania jednostek na wrogim terytorium.

Testy Kolizji Robotów i Systemu Walki

Po wprowadzeniu funkcji przewijania tury, wprowadzony został system możliwych kolizji. Został on przetestowany poprzez wstępne ustawienie wrogich sobie jednostek i wprowadzenie ich na tor kolizyjny w różnych scenariuszach. Po naniesieniu niewielkich poprawek, funkcja spełniała swoje zadanie.

Testy Płynności Rozgrywki

Ze względu na niewielką ilość modeli 3D i ich prostotę, nawet po ustawieniu wielu jednostek na mapie, gra działa płynnie.

Ogólne Testy Funkcjonalności

Poszczególne funkcje były testowane w trakcie pracy poprzez dokładną analizę ich przebiegu i zmiany wartości danych zmiennych podczas debugowania. Wykonywane zostały przypadki, w których jako dane wejściowe były podawane różne wartości, aby sprawdzić zachowanie konkretnych funkcji w danych sytuacjach.