Implementacja silnika microRTS na platformę CodinGame

(Implementation of microRTS engine for CodinGame platform)

Mikołaj Motak

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

19 lutego 2024

Streszczenie

Praca przedstawia implementację silnika microRTS na platformę CodinGame. Jest to projekt wzorujący się na aplikacji Santiago Ontañóna, napisanej w celu stworzenia platformy naukowej do testowania algorytmów sztucznej inteligencji przy użyciu gier strategii czasu rzeczywistego. Idea ta wpasowuje się w koncepcję CodinGame, która udostępnia przegląd problemów logicznych i prostych gier, które mają służyć nauce programowania – w szczególności programowania sztucznej inteligencji. Interfejs i mechanika gry zostały dostosowane do platformy CodinGame. W pracy zostały opisane szczegóły implementacji, ale również specyfikacja samego projektu oraz zasady gry.

This thesis describes an implementation of microRTS engine for CodinGame platform. It is a project inspired by application written by Santiago Ontañón – made to create a research platform to test AI algorithms using real-time strategy games. Idea fits the concept of the CodinGame, which offers range of logic puzzles and simple games, meant to help hone programming skills – specifically those revolving around artificial intelligence. User interface and game mechanics were adjusted to the CodinGame. This paper describes implementation details, but also other specifications about the project and rules of the game.

Spis treści

Rozdział 1.

Wprowadzenie

1.1. Cel projektu

Celem tej pracy jest zaimplementowanie wariantu gry microRTS na platformę CodinGame. To projekt, który ma służyć do nauki pisania sztucznej inteligencji, w szczególności botów do gier bazujących na znajdowaniu drogi, podejmowaniu decyzji oraz zarządzaniu zasobami. Jest to reprezentacja połączenia zarówno idei microRTS Santiago Ontañóna [1], jak i idei stojącej za platformą CodinGame, czyli poszerzenie ogólnodostępnych zasobów edukacyjnych. Sama implementacja ma również służyć pokazaniu wdrożenia gry na dane środowisko programistyczne; przedstawieniem wyzwania, jakim jest implementacja projektu w specjalistycznym i ograniczonym systemie.

1.2. Strategia czasu rzeczywistego

MicroRTS to uproszczona wersja gatunku RTS (real-time strategy; strategii czasu rzeczywistego). Charakterystyką takich gier jest zbieranie zasobów, budowanie baz, trenowanie jednostek oraz pośrednia kontrola jednostkami. Nie jest to jedyny gatunek gier, który ma te cechy, ale w odróżnieniu od innych gier taktycznych, symulatorów budowania oraz symulatorów zarządzania – gry RTS zazwyczaj łączą wszystkie te elementy. RTS są podgatunkiem gier strategicznych, które są rozgrywane jednocześnie przez wszystkich graczy, w czasie rzeczywistym. To oznacza, że nie ma podziału na tury, a wszystkie fazy gry (jak podejmowanie decyzji czy wykonywanie akcji) dzieją się równocześnie dla wszystkich graczy. Przykładami takich gier są: StarCraft, Frostpunk, Age of Empires, WarCraft czy Stellaris.

Wersja mikro takich gier charakteryzować powinna się połączeniem wszystkich tych cech, bez zbędnej ilości dodatkowych elementów rozgrywki, skomplikowanych zasad, jednostek, budynków oraz metod. Sama rozgrywka powinna się natomiast odbywać na uproszczonej, przejrzystej arenie; oferując minimalne opcje zarządzania,

zbieractwa oraz walki. Przewagą takiej uproszczonej implementacji, jest szybkość testowania pomysłów oraz oprogramowania, przed przejściem do poważniejszych projektów i pełnych wersji gier. Deterministyczna natura rozgrywki pozwala natomiast na łatwiejsze zrozumienie oraz rozpatrywanie problemu.

1.3. MicroRTS, Farama Foundation i zawody podczas konferencji CoG

Oryginalna implementacja microRTS [2] została stworzona przez Santiago Ontañóna (obecnie starszego pracownika naukowego w Google Research oraz profesora nadzwyczajnego na Drexel University). Stworzył on symulator uproszczonych gier strategii czasu rzeczywistego, który umożliwia testowanie botów, jak i ręczną rozgrywkę zgodną z zasadami gatunku. Program pozwala na dostosowywanie większości aspektów i zasad gry. Obecnie system ten jest dostępny jako część Farama Foundation – organizacji non-profit, dążącej do stworzenia i utrzymywania otwartych (open-source) narzędzi do nauki programowania [3].

System microRTS Ontañóna był również od 2017 roku używany do organizacji zawodów sztucznej inteligencji podczas konferencji CoG [4] (Conference on Games). Konkurs microRTS został stworzony, aby motywować do badań nad podstawowymi pytaniami badawczymi leżącymi u podstaw rozwoju sztucznej inteligencji w grach RTS, przy jednoczesnym minimalizowaniu ilości prac inżynieryjnych wymaganych do wzięcia udziału.

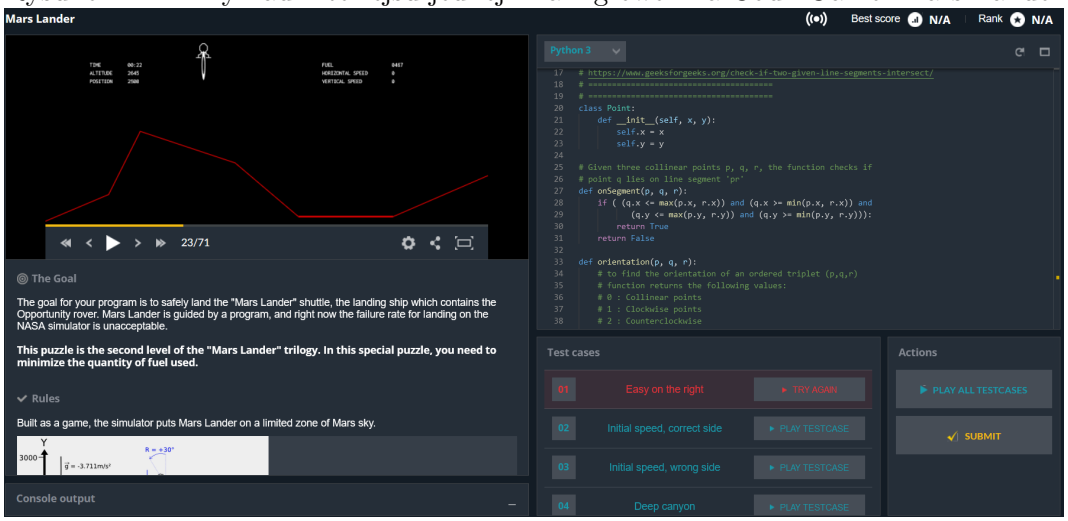
Zawody [5] zorganizowane są wokół wariantów gry – każdy wariant nakierowany na jedno kluczowe wyzwanie badawcze z dziedziny sztucznej inteligencji w grach RTS (uczestnicy mogą zgłosić się do jednej lub więcej kategorii). Na przełomie lat 2017-2021, były to zazwyczaj trzy warianty rozgrywki (klasyczny, niedeterministyczny oraz o ograniczonej widoczności), a w roku 2023 był to tylko wariant deterministyczny – skupiający się na problemie dużej przestrzeni stanów oraz czynnikach rozgałęziających (wynikających z pełnej widoczności podczas rozgrywki).

Zwycięzca każdego konkursu wyłaniany jest w drodze turnieju wedle zasad „każdy z każdym", podczas którego rozgrywki odbywają się na zestawie map, a programy zawodników walczą ze wszystkimi pozostałymi uczestnikami oraz kilkoma botami dodanymi przez organizatorów. Turniej spotkał się z pozytywnym odzewem uczestników konferencji i w każdej edycji gromadzi w sobie do kilkunastu zawodników.

1.4. CodinGame

CodinGame [6] jest platformą edukacyjną, której celem jest rozwijanie zdolności programowania jej użytkowników, ze szczególnym naciskiem na poznawanie metod oraz przykładów implementacji algorytmów sztucznej inteligencji. Strona oferuje szeroki zasób łamigłówek logicznych (rys. 1.1) oraz prostych gier, których struktura i zasady mają na celu przedstawienie problemów programistycznych – których postać powinna inspirować do poznawania istniejących algorytmów z kategorii odnajdowania drogi, optymalizacji, zarządzania, przetwarzania danych, kryptografii oraz uczenia maszynowego i wielu innych. Problemy różnią się poziomem trudności, co pozwala na ugruntowanie umiejętności pisania kodu w dostępnych językach programowania. CodinGame oferuje również system osiągnięć oraz różnego rodzaje wydarzenia jak na przykład zawody pisania botów, inspirujące do dalszego korzystania z platformy.

Rysunek 1.1: Przykład interfejsu jednej z łamigłówek na CodinGame: Mars Lander



Rozdział 2.

Zasady gry

2.1. Definicja gatunku a specyfika platformy CodinGame

Gry RTS to podgatunek gier strategicznych, w których wszystkie elementy rozgrywki odbywają się jednocześnie, w czasie rzeczywistym. Nie ma podziałów na tury, czy fazy, a podejmowanie decyzji oraz wydawanie rozkazów jest przeprowadzane przez wszystkich graczy w tym samym czasie.

CodinGame jest platformą edukacyjną udostępniającą problemy informatyczne, gry i łamigłówki, dla których użytkownik pisze programy oraz boty mające na celu rozwiązanie postawionych problemów (jak np. wygranie gry). Specyficznym elementem tej platformy, zwłaszcza w grach jest turowa natura rozgrywki. Gracze co turę otrzymują aktualne dane na temat stanu gry i na ich podstawie odpowiadają przy użyciu zdefiniowanych rozkazów. Taka forma rozpatrywania problemów ułatwia ich zrozumienie oraz ogranicza ilość informacji, jaką należy przetworzyć w danym momencie.

Jak można wywnioskować z obu powyższych twierdzeń gry RTS niekoniecznie, na pierwszy rzut oka, wpisują się w styl prezentowania problemów na CodinGame. Jednakże możliwe jest wykorzystanie systemu turowego do zasymulowania rozgrywki w czasie rzeczywistym. Czas jest wymiarem podzielnym – w prawdziwym świecie mierzymy upływ czasu przy użyciu między innymi sekund, minut, godzin. Pierwszym krokiem naszego rozwiązania jest zauważenie, że tury gry są tożsame z jednostkami czasu. Podczas jednej tury oraz podczas jednej sekundy jesteśmy w stanie wykonać tylko pewną ilość akcji. I ta wartość jest niezmienna ze względu na ograniczenia nałożone przez nas przez grę oraz prawa fizyki (np. możemy tylko wykonać jeden krok podczas jednej tury/sekundy).

Innym problemem jest wykonywanie działań przez wszystkich graczy jednocześnie. RTS wymaga, aby gracze nie czekali na ich kolej, aby wydać kolejne polecenia, jednakże CodinGame przekazuje informacje o stanie gry każdemu graczowi jeden po drugim. Rozwiązaniem tego problemu jest traktowanie czasu na przekazywanie

danych, obliczenia graczy oraz odbieranie od nich rozkazów jako wydarzenia, które dzieje się przed każdą turą rozgrywki. W trakcie tej fazy stan gry nie jest zmieniany, więc z punktu widzenia gry ta faza dzieje się poza czasem symulacji, Jeżeli więc obaj gracze otrzymują informacje i wykonują obliczenia momentalnie – to wykonują te działania jednocześnie.

Ostatnim problemem jest rozpatrywanie rozkazów obu graczy w tym samym czasie. Program może tylko przetwarzać każde polecenie jedno po drugim, dlatego w naszej symulacji, musimy zapewnić, aby żaden z graczy nie był faworyzowany. Sytuacja komplikuje się tylko, gdy gracze wpływają na siebie lub starają się wykonać wykluczające siebie nawzajem polecenia. W przypadku gier RTS do takich sytuacji dochodzi, gdy gracze wykonują atak oraz gdy poruszające się jednostki blokują sobie drogę (np. chcąc wykonać krok na to samo pole). W przypadku obrażeń sprawa jest prosta – wystarczy zapewnić, aby oba polecenia ataku zostały wykonane (gdy obaj gracze atakują jednocześnie – obaj powinni otrzymać obrażenia).

W przypadku pozostałych konfliktowych sytuacji jednym z rozwiązań byłoby zablokowanie sprzecznych poleceń (gracze starają się ruszyć na to samo pole, więc nikt nie porusza się na to pole), jednakże w takich sytuacjach możemy doprowadzić do zbyt statycznej rozgrywki, podczas której nikt nie może wykonać swoich ruchów. Innym rozwiązaniem byłoby wybieranie na zmianę, która jednostka może poruszyć się w danej turze. Trzecim rozwiązaniem jest wykonanie tylko jednego losowo wybranego polecenia. Minusem tego jest utracenie symetryczności pomiędzy graczami, ale losowość zapewnia sprawiedliwe rozpatrywanie rozgrywki. Takie rozwiązanie również inspiruje do tworzenia przez graczy bardziej zaawansowanych strategii, które starają się ominąć sytuacje, w których los decyduje o wyniku. Ostatnim rozwiązaniem byłoby umożliwienie okupowania przez dwie jednostki tego samego pola, jednakże w takim przypadku gra traci na przejrzystości i może doprowadzić do sytuacji, gdzie zbyt wiele jednostek okupuje to samo pole.

Pierwsze trzy rozwiązania konfliktów są zaimplementowane w wersji Santiago Ontañóna. Wariant blokowania oraz wariant losowy były używane podczas turniejów programistycznych microRTS organizowanych w trakcie konferencji CoG. Od 2023 roku tylko metoda blokowania jest używana (ze względu na mniejsze zainteresowanie niedeterministycznymi wersjami turnieju). Jako że jest to również domyślna metoda w implementacji Ontañóna – w implementacji na CodinGame konflikty są również rozpatrywane poprzez zablokowanie.

Oprócz ograniczeń technicznych platforma CodinGame ma pewną specyfikę co do tego, w jaki sposób każda gra jest prezentowana użytkownikowi piszącemu boty. Strona składa się z trzech głównych sekcji – ekranu wizualizacji, opisu gry oraz edytora kodu. Nasza implementacja oprócz samego symulowania gry, przetwarzania danych oraz przesyłania wyników do graczy, musi również zapewnić, aby użytkownik miał dostęp do graficznej reprezentacji każdej tury rozgrywki. Należy więc stworzyć taką implementację, która zawiera w sobie uproszczoną formę RTS, ale również nie

jest ona zbyt skomplikowana do wyrysowania na platformie CodinGame.

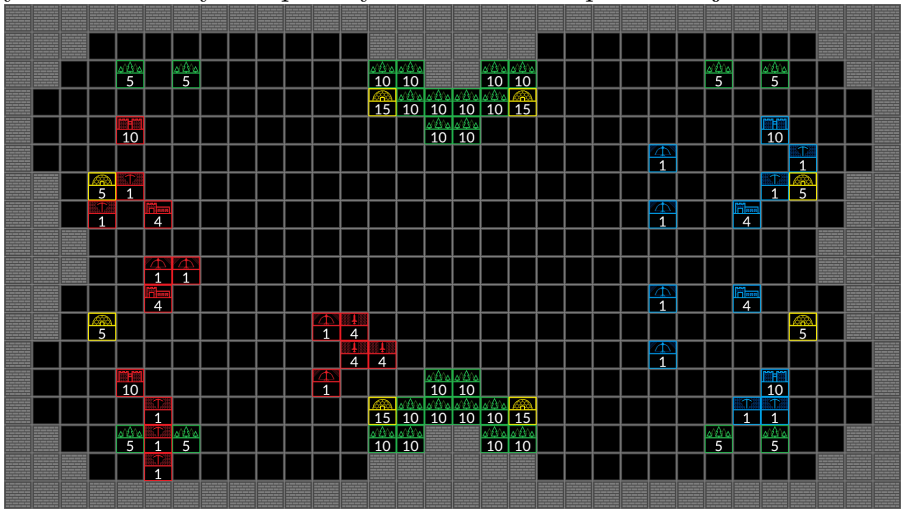
2.2. Podstawy rozgrywki

Zaimplementowana wersja microRTS jest odgrywana na mapie podzielonej na pola (rys. 2.1). Taka zuniformizowana reprezentacja planszy pozwala skupić się na samej rozgrywce, bez potrzeby rozpatrywania czysto kosmetycznych elementów jak np. rozmiar jednostek. Taka forma przypomina planszowe gry taktyczne jak np. szachy i pozwala na prostsze zrozumienie stanu gry. Każde pole może być zajęte w tym samym momencie tylko przez jedną jednostkę (niepuste pola należące do graczy jak jednostki bojowe oraz budynki, ale również neutralne pola jak ściany oraz kopalnie).

Na początku każdej tury gracze otrzymują listę wszystkich niepustych pól na mapie z informacją o ich pozycji, właścicielu, typie jednostki oraz ilości jej punktów życia. Każdy gracz rozpoczyna rozgrywkę z określoną ilością materiałów (które również można znaleźć na mapie, a które należy wydobyć). Z tych materiałów gracze mogą tworzyć jeszcze więcej innych jednostek bojowych oraz budynków.

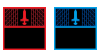
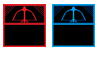
Po otrzymaniu i przetworzeniu danych gracze przesyłają rozkazy dla swoich jednostek. Wszystkie rozkazy muszą być napisane zgodnie z regułami gry oraz składnią rozkazów. Niepoprawne polecenia powodują automatyczną porażkę gracza. Jeżeli rozkaz spełnia wymagania, ale nie jest możliwy do wykonania z innych powodów (np. inna jednostka blokująca drogę) – to taki rozkaz zakończy się niepowodzeniem, bez przegranej gracza. Każda jednostka ma tylko określone polecenia, które może wykonać. Wszystkie akcje mogą być wykonywane we wszystkich 8 kierunkach na planszy. Jednostki są tylko ograniczone przez ilość wolnego miejsca, inne jednostki, własny zasięg oraz wielkość kroku, jaki mogą zrobić podczas jednej tury. Mogą się tylko poruszać do pól, które są puste lub które zostaną opuszczone do końca tury.

Rysunek 2.1: Przykład planszy microRTS w implementacji na CodinGame



2.3. Jednostki i budowle

Wszystkie jednostki mogą zostać sklasyfikowane ze względu na ich typ oraz ich właściciela:

- Pola neutralne – są elementami areny, które służą do urozmaicenia rozgrywki przez ograniczenie dostępu do materiałów oraz utrudnienie poruszania się jednostek graczy. Nie należą do żadnego z graczy. Oprócz pustych pól w zaimplementowanym wariancie gry można wyróżnić trzy neutralne typy pól:
  -  Ściana – główna fizyczna przeszkoda na mapie. W wybranym wariancie rozgrywki, wszystkie ściany są niezniszczalne.
  -  Kopalnia – źródło materiałów, z którego robotnicy wydobywają Złoto, aż do wyczerpania zapasów. Znika po wydobyciu wszystkiego.
  -  Las – źródło materiałów, z którego robotnicy wydobywają Drewno, aż do wyczerpania zapasów. Znika po wydobyciu wszystkiego.
- Budynki – są przykładem statycznych pól należących do graczy. Nie mogą się poruszać lub atakować, ale mogą trenować jednostki.
  -  Zamek – kluczowy budynek dla każdego z graczy. Pozwala trenować Robotników. Gracz, który utraci wszystkie zamki – przegrywa.
  -  Koszary – miejsce, które pozwala wytrenować bardziej zaawansowane jednostki bojowe: Lekkie, Ciężkie i Dystansowe.
- Jednostki (bojowe) – to bardziej dynamiczne pola należące do graczy.
  -  Robotnik – najbardziej uniwersalna jednostka. Główną jej użytecznością jest zbieranie zasobów oraz budowanie. Robotnicy mogą atakować, ale nie są zbyt wytrzymali, silni, czy szybcy, ich zasięg również jest ograniczony.
  -  Lekka – zaawansowana jednostka bojowa. Bardziej wytrzymała i silniejsza niż zwykli Robotnicy. Jej główną cechą jest to, że może poruszać się dwa pola na turę (jest szybsza od innych jednostek).
  -  Ciężka – najsilniejsza i najbardziej wytrzymała jednostka w grze, droższa w produkcji od innych jednostek.
  -  Dystansowa – słaba jednostka dalekiego zasięgu. Jej główną cechą jest duży zasięg ataku wynoszący trzy pola.

2.4. Tura gry

Podczas każdej tury gracze otrzymują szczegółowe informacje o każdym niepustym polu na mapie oraz o ilości surowców, jakie posiadają (tab. 2.1). Każdy z graczy przeprowadza własne obliczenia i przesyła grze jedną linię tekstu zawierającą listę rozkazów oddzielonych średnikami (tab. 2.2). Dla każdego rozkazu gracz musi podać dwie pary współrzędnych (dla jednostki oraz celu). np. „TRAIN 1 1 3 4 WORKER", „MOVE 2 3 2 4", „HARVEST 3 3 4 3". Następnie arbiter gry sprawdza poprawność przesłanych rozkazów oraz wykonuje je wedle kolejności: budowanie → trening jednostek → poruszanie się → zbieranie surowców → atak. Każdy rozkaz poruszania się, który nie może być od razu przeprowadzony zostaje przerzucony na koniec kolejki fazy ruchu. Zabieg ten jest powtarzany do momentu aż kolejka zostanie opróżniona lub żadna z jednostek nie może się już więcej poruszyć. Na koniec fazy ruchu wszystkie rozkazy budowania lub treningu, które nie były możliwe wcześniej do wykonania, są ponownie rozpatrywane. Sprzeczne rozkazy o tym samym priorytecie kolejności nie są wykonywane. Gdy dwie jednostki atakują siebie nawzajem – obie otrzymują obrażenia. Na sam koniec arbiter czyści mapę z wszelkich pokonanych jednostek i opróżnionych źródeł surowców, a następnie sprawdza, czy którykolwiek z warunków końcowych został spełniony (czy któryś z graczy utracił wszystkie zamki lub gra osiągnęła limit tur).

Tabela 2.1: Dane jednostek

Nazwa jednostki	Typ	Życie	Zasięg	Siła	Krok	Koszt	
Zamek	0	10	3	0	0	5 złota	5 drewna
Koszary	1	4	2	0	0	5 drewna	
Robotnik	2	1	1	1	1	1 złota	
Lekka	3	4	1	2	2	1 złota	1 drewna
Ciężka	4	8	1	4	1	2 złota	1 drewna
Dystansowa	5	1	3	1	1	1 złota	1 drewna

Tabela 2.2: Rozkazy

Nazwa jednostki	Rozkaz	Cel	Składnia
Zamek	trening	Robotnik	TRAIN WORKER
Koszary		Lekka	TRAIN LIGHT
		Ciężka	TRAIN HEAVY
		Dystansowa	TRAIN RANGED
Robotnik	budowanie	Zamek	BUILD CASTLE
		Koszary	BUILD BARRACKS
	zbieranie	Kopalnia	HARVEST
		Las	
Robotnik Lekka Ciężka Dystansowa	ruch	Puste pole	MOVE
	atak	Jednostki przeciwnika	ATTACK

2.5. Implementacja Santiago Ontañóna

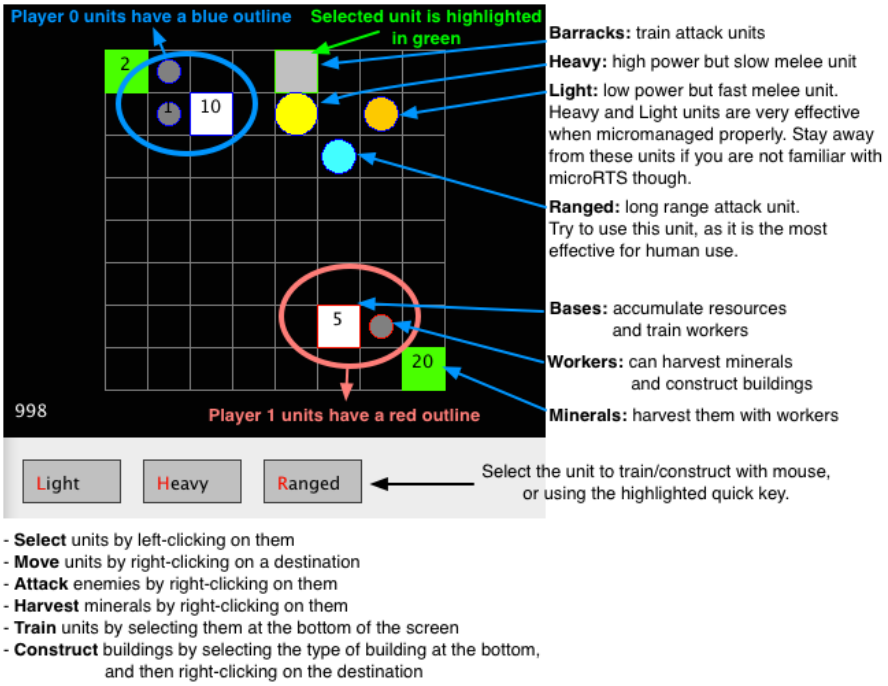
MicroRTS Santiago Ontañóna ma formę aplikacji okienkowej. Jest symulatorem gier z tego gatunku, który pozwala na bardzo szczegółową personalizację całej rozgrywki (od tworzenia różnorodnych scenariuszy, edycji map, po zmiany statystyk jednostek oraz algorytmów rządzących decyzjami botów). Uproszczona forma mapy oraz ogólna implementacja poszczególnych rozkazów i jednostek są niemal tożsame z implementacją na CodinGame omawianą w tym opracowaniu. Jednakże pewne elementy wersji Ontañóna musiały zostać zmienione, by wpasować się w specyfikę CodinGame.

Znaczącą różnicą jest fakt, że oryginalna aplikacja pozwala na rozgrywkę używając botów sztucznej inteligencji, jak i również posiada interfejs umożliwiający bezpośrednią kontrolę przy użyciu myszki i klawiatury (rys. 2.2). Gracze mogą zaznaczać jednostki oraz używać skrótów klawiszowych, aby wydawać poszczególne rozkazy. Cała rozgrywka dzieje się w czasie rzeczywistym – każda czynność ma wyznaczony czas wykonywania. W wersji CodinGame cały ten proces został uproszczony i ujednolicony, aby wpasować się w naturę platformy. W szczególności w turowy aspekt przetwarzania danych. Gracze mogą decydować o wszystkich jednostkach równocześnie, choć wciąż są ograniczeni wykonywaniem jednej akcji na turę dla każdej jednostki. Wynika to z ograniczeń czasowych, jakie narzuca na nas CodinGame (rozpatrywanie rozgrywki poprzez pojedyncze rozkazy mogłoby doprowadzić do przekroczenia limitów czasowych).

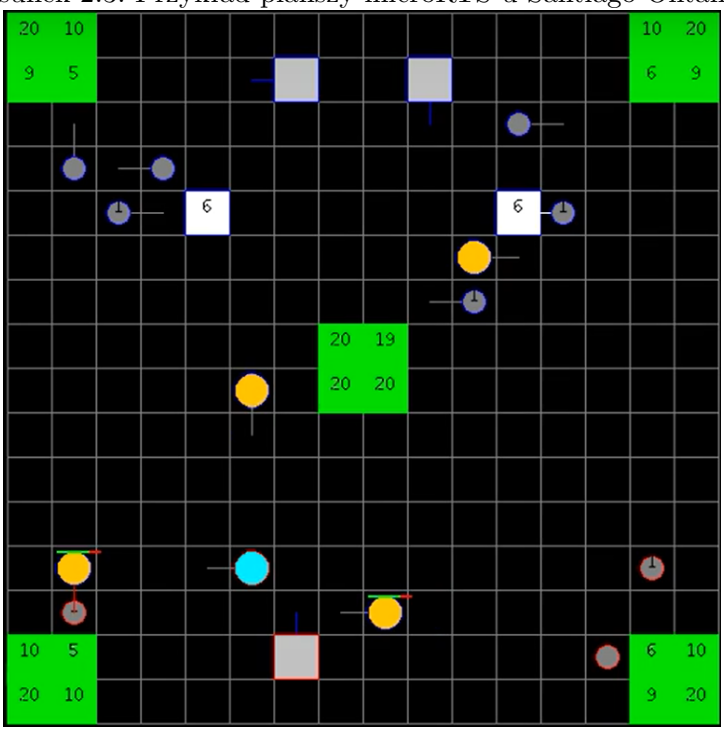
Innym elementem odróżniającym obie implementacje jest zbiór map, w które gracze mogą zagrać. MicroRTS Santiago Ontañóna posiada kolekcję gotowych map (rys. 2.3) – wiele z nich stworzonych na potrzebę turniejów odbywających się pod-

czas konferencji CoG. Implementacja na CodinGame natomiast generuje mapy przed każdym pojedynkiem, opierając się na agregacji ograniczonej dyfuzją (proces wynikający z Ruchów Browna, w którym cząsteczki wykonujące chaotyczne, losowe ruchy skupiają się w zbiory).

Rysunek 2.2: Charakterystyka planszy microRTS u Santiago Ontañóna



Rysunek 2.3: Przykład planszy microRTS u Santiago Ontañóna



Rozdział 3.

Implementacja

3.1. Technologie

Implementacja gry napisana została w większości przy użyciu języka Java. Do wizualizacji rozgrywki silnik gry wykorzystuje język JavaScript, a w szczególności bibliotekę PixiJS, rekomendowaną przez platformę CodinGame.

3.2. Struktura

Cały projekt implementacji microRTS na CodinGame, można rozbić na trzy główne zbiory plików, w zależności od ich celu oraz interaktywności z platformą CodinGame (rys. 3.1).

3.2.1. Ustawienia

- Boss.py – prosty bot napisany w języku Python, wykorzystujący przeszukiwanie wszerz [7] jako główny algorytm wyszukiwania drogi. Jest to podstawowy przeciwnik udostępniony na platformie, z którym użytkownicy muszą się zmierzyć. Jego głównym celem jest zaprezentowanie funkcjonalności projektu.

- config.ini – plik konfiguracyjny ustalający typ gry (w tym przypadku typ wieloosobowy) oraz ilość możliwych graczy (w tym wariancie tylko dwóch).

- statement_en.html – plik z opisem rozgrywki. Jest to główna reprezentacja projektu z dokładnym opisem specyfikacji i zasad gry, do której mają dostęp użytkownicy CodinGame podczas korzystania z platformy. Plik zawiera wszelkie informacje potrzebne do zrozumienia problemu oraz sposobu komunikacji z grą (jakie dane otrzymujemy, jakie zwracamy, limity czasowe oraz szczegółowe informacje o jednostkach).

- stub.txt – plik napisany w pseudokodzie zawierający szkielet bota. CodinGame tłumaczy ten pseudokod do jednego z kilkunastu dostępnych języków programowania oferowanych na platformie.

3.2.2. Zasoby

- assets – folder zawierający wszystkie pliki graficzne używane do wizualizacji projektu.
- config.js – plik konfiguracyjny importujący listę modułów użytych w projekcie. Definiuje również podstawowe zmienne graficzne – jak np. kolory graczy.
- demo.js – demonstracyjna animacja, pokazująca rozgrywkę oraz tytuł gry. Plik ten jest odtwarzany na ekranie służącym do wizualizacji rozgrywki zaraz po otworzeniu gry na CodinGame. Jest swoistą introdukcją do projektu i pierwszą rzeczą, jaką zobaczą użytkownicy.

3.2.3. Silnik

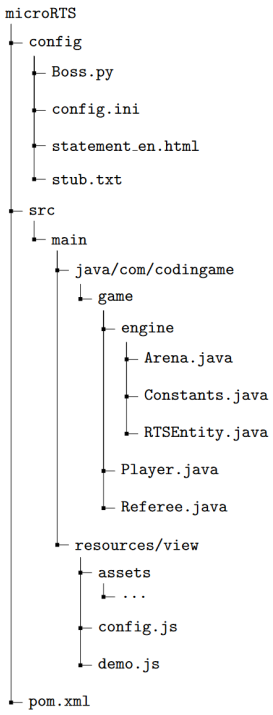
- Referee.java – arbiter rozgrywki. Główna klasa projektu, która inicjuje cały projekt, przesyła i odbiera dane od graczy, wykonuje wszelkie obliczenia, odwołując się do innych klas oraz jest głównym mostem pomiędzy użytkownikiem, modułami, a silnikiem CodinGame. Zawiera również funkcje aktualizujące graficzną reprezentację projektu.
- Player.java – klasa rozszerzająca obiekt gracza zdefiniowany w silniku CodinGame. Umożliwia odbieranie danych (rozkazów) od poszczególnych graczy oraz zawiera listę surowców, którą dany gracz posiada.
- Arena.java – zawiera wszystkie struktury, w których zapisany jest aktualny stan rozgrywki – zarówno status jednostek, jak i również graficzna reprezentacja mapy (pliki graficzne oraz tekst używane do wizualizacji). Jest to również klasa, w której znajdują się wszystkie funkcje używane do inicjowania oraz edycji stanu gry.
- Constants.java – plik zawierający wszystkie zmienne wartości definiujące wariant gry. Od rozmiaru mapy, przez limity czasowe, po definicje jednostek.
- RTSEntity.java – klasa obiektu RTSEntity czyli jednostek używanych przez graczy. Zawiera współrzędne pozycji na mapie oraz listę rozkazów dana jednostka otrzymała od gracza.

3.2.4. Inne

- pom.xml – obiektowy model projektu, zawiera listę potrzebnych wersji, zależności oraz modułów.

- resources – główny folder z zasobami, oprócz już wymienionych elementów, zawiera również pliki tekstowe definiujące wygląd map. Jest główną lokalizacją w projekcie dla wszelkich dodatkowych plików graficznych oraz innych zasobów.

Rysunek 3.1: Uproszczona wersja hierarchii plików projektu



3.3. Generowanie map

Implementacja na CodinGame posiada prosty generator map, opierający się na agregacji ograniczonej dyfuzją (diffusion-limited aggregation – DLA) [8]. Jest to model stworzony przez Thomasa Wittena oraz Leonarda Sandera w 1981, mający zastosowanie do opisu agregacji w każdym systemie, w którym dyfuzja jest głównym środkiem transportu. DLA można zaobserwować w wielu układach, takich jak galwanizacja, osady mineralne i wyładowanie elektryczne. Cząsteczki osadu omawiane w tym procesie wykonując chaotyczne, losowe ruchy skupiają się w zbiory, o kształcie losowych fraktali znanych również jako drzewa Browna. Proces ten jest możliwy do zaimplementowania [9] – tworząc zbiór połączonych punktów o unikatowym kształcie nadającym się jako podstawa dla map do gier.

Pierwszym krokiem implementacji tego modelu jest wyznaczenie jądra naszej konstrukcji. Następnie umieszczany cząsteczkę osadu w losowym miejscu na mapie. Kolejnym krokiem jest losowe poruszanie się cząsteczki, aż natrafi ona na jądro. Proces ten jest powtarzamy dla kolejnych cząsteczek, aż utworzą one zadowalającą strukturę osadową. W naszym przypadku cząsteczkami są puste pola, a końcowy fraktal utworzy szkielet połowy naszej mapy. W implementacji na CodinGame –

generator wyznacza brzeg mapy jako jądro struktury, w ten sposób po zakończonym procesie możemy wykonać odbicie lustrzane, które stworzy symetryczną mapę rozgrywki. Na koniec symetrycznie dodajemy budynki, jednostki oraz źródła zasobów (każdy z tych elementów ma w implementacji zdefiniowaną minimalną oraz maksymalną liczbę wystąpień, dla każdej mapy – losujemy inną wartość z tego zakresu).

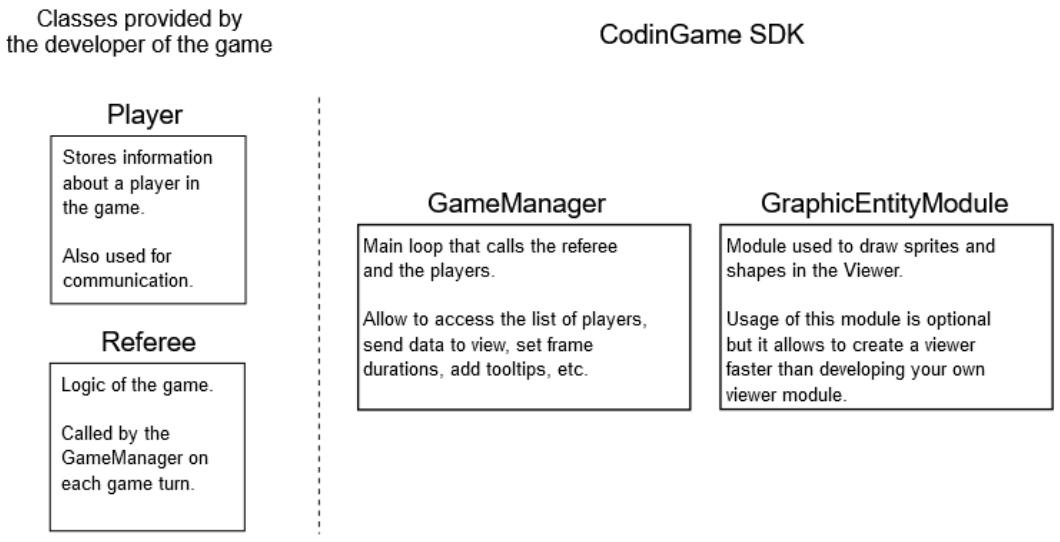
3.4. CodinGame SDK

CodinGame oferuje silnik (rys. 3.2), na którym oparty został cały projekt omówiony w tej pracy. Szeroka dokumentacja [10] udostępnia zarówno wszelkiego rodzaju warianty rozgrywki, jak i różne moduły graficzne i logiczne, które stanowią stabilną bazę do tworzenia gier i łamigłówek. Głównymi klasami CodinGame SDK są:

- GameManager – klasa zajmuje się przebiegiem gry, przeprowadzaniem kolejnych tur oraz obliczaniem każdej klatki wizualizacji rozgrywki. Zapewnia wiele metod użytkowych obsługujących inne klasy, moduły i instancje projektu. Tworzy główną pętlę gry, w której wywołuje arbitra rozgrywki oraz wszystkich graczy.

- GraphicEntityModule – moduł zajmuje się wyświetlaniem i animacją elementów graficznych podczas wizualizacji gry. Używany przez tworzenie kształtów, tekstów oraz ikon i przypisywanie ich stanów do określonych klatek/momentów wizualizacji.

Rysunek 3.2: Struktura projektu na CodinGame [11]



Dokumentacja zawiera również przykłady gier napisanych w tym systemie. Gra wieloosobowa "Pong Game"[12] została użyta jako jedno z odniesień i baz dla stworzenia implementacji microRTS na CodinGame.

3.5. Społeczność

CodinGame ma społeczność aktywnie udzielającą się w procesie tworzenia nowych kontrybucji na platformie. Po udostępnieniu nowego projektu i ustawieniu jego widoczności jako „Work in Progress" użytkownicy CodinGame mają możliwość testowania oraz komentowania nowej pracy. Jest to pierwszy etap oceny nowego projektu, dający bezpośredni kontakt z grupą docelową.

3.6. Uruchomienie programu

Projekt można uruchomić poprzez platformę CodinGame [13]. Umożliwia to zapoznanie się z zasadami rozgrywki, napisaniem bota w różnych językach programowania, oraz testowania ich przeciwko wbudowanemu botowi. Platforma udostępnia widok całej symulowanej rozgrywki z możliwością przewijania przez tury/klatki. Kod źródłowy projektu można również znaleźć na platformie GitHub [14].

Rozdział 4.

Podsumowanie

Celem projektu było zaimplementowanie wariantu gry microRTS na platformę CodinGame inspirowanego aplikacją stworzoną przez Santiago Ontañóna. Głównym zadaniem było dostosowanie implementacji pod specyfikę CodinGame. Projekt pozwala na dalszy rozwój przez dodanie dodatkowych funkcjonalności do bazowej wersji gry. Do przykładów należy m.in, eksploracja poważniejszych algorytmów sztucznej inteligencji poprzez zaimplementowanie ich jako domyślnego bota do gry. Innym kierunkiem mogłoby być rozszerzenie funkcjonalności o inne aspekty gier strategii czasu rzeczywistego, jak np. rozszerzenie elementu ekonomicznego, zarządzania oraz rozwoju.

Bibliografia

[1] Santiago Ontañón, https://sites.google.com/site/santiagoontanonvillar/

[2] Santiago Ontañón (2013) The Combinatorial Multi-Armed Bandit Problem and its Application to Real-Time Strategy Games, In AIIDE 2013. pp. 58 - 64.

[3] Oryginalny silnik microRTS, https://github.com/Farama-Foundation/MicroRTS

[4] Conference on Games, https://2023.ieee-cog.org/

[5] microRTS Competition, https://sites.google.com/site/micrortsaicompetition

[6] CodinGame, https://www.codingame.com

[7] Coder Space, "Pathfinding Algorithms in Python. BFS, Dijkstra, A Star", https://www.youtube.com/watch?v=abHftC1GU6w

[8] DLA, https://en.wikipedia.org/wiki/Diffusion-limited_aggregation

[9] Implementacja DLA, https://www.roguebasin.com/index.php/Diffusion-limited_aggregation

[10] Silnik CodinGame, https://github.com/CodinGame/codingame-game-engine

[11] Dokumentacja SDK CG, https://www.codingame.com/playgrounds/25775/codingame-sdk-documentation

[12] Przykład gry multiplayer, https://github.com/CodinGame/game-pong

[13] Implementacja microRTS na CodinGame, https://www.codingame.com/contribute/view/83868247f4b740d06bc96610a701291982a22

[14] Implementacja microRTS na CodinGame, https://github.com/motakmikolaj/microRTS