Implementacja Gry Programistycznej Roguelike na platformę CodinGame

(Roguelike Game Implementation for CodinGame platform)

Kamila Adamczyk

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

1 września 2019

Streszczenie

Praca jest częścią projektu mającego na celu stworzenie gry typu roguelike, do której możliwe będzie pisanie programów w nią grających. Gra polega na przemierzaniu bohaterem labiryntu w poszukiwaniu wyjścia, przy równoczesnym zbieraniu skarbów i walce z potworami. Celem gry jest opublikowanie jej na platformie CodinGame, służącej do nauki i rozwijania umiejętności programowania, głównie poprzez udział w zawodach pisania botów do prostych gier. W pracy opisane są wymogi, jakie powinien spełniać taki program oraz opis jego tworzenia i implementacji.

This thesis is a part of the project aimed at creating a roguelike game that can be played by bots. In this game the bot has to find its way out of the dungeon, while collecting treasures and fighting enemies along the way. The goal of this project is to release the game at CodinGame. This platform allows the developers to learn and advance their coding skills, mainly through participating in the contests based on writing bots to simple games. This thesis describes what requirements such project has to fulfill and how they were implemented.

Spis treści

Rozdział 1.

Wstęp

1.1. Cel pracy

Poniższa praca jest częścią projektu mającego na celu stworzenie i opublikowanie gry typu roguelike na platformie CodinGame. Gra rozgrywana jest turami, a jej celem jest dotarcie do wyjścia z labiryntu, zbierając po drodze skarby i walcząc z przeciwnikami, aby uzyskać jak najwyższy wynik.

1.2. Gry roguelike

Roguelike to gatunek gier komputerowych, którego korzenie wywodzą się z gry „Rogue" z lat 80., w której gracz przemierzał proceduralnie generowane poziomy walcząc z potworami oraz zbierając broń, mikstury i magiczne zwoje. Gry, które zapożyczały ten koncept rozgrywki, zaczęto nazywać roguelike. Do cech charakterystycznych takich gier należą m.in:

- permanentna śmierć – gracz nie ma możliwości wczytania gry po śmierci swojej postaci,
- rozgrywka podzielona na tury,
- strata zasobów (np. wytrzymałości) w każdej turze, wymuszająca zarządzanie zasobami i rozsądne poruszanie się po planszy,
- mgła wojny – gracz widzi tylko na pewną odległość od siebie, a mapa jest stopniowo odsłaniana wraz z przemieszczaniem się po planszy. Obszar, na którym gracz był wcześniej, może pozostać odsłonięty lub z powrotem zakryty.

1.3. Roguelike a sztuczna inteligencja

Sztuczna inteligencja do przechodzenia gier roguelike była badana m.in. w pracy „Evolving Personas for Player Decision Modeling"[1], która bazowała na grze „MiniDungeons". Zostały w niej porównane algorytmy ewolucyjne tworzenia botów z algorytmem Q-learning. Problem ten, razem z proceduralnym generowaniem podziemi, został opisany w pracy „Roguelike Games as a Playground for Artificial Intelligence – Evolutionary Approach"[2], na potrzeby której stworzono platformę do prostego pisania botów dla gry „Desktop Dungeon", w której dodatkowo dostępne były różne klasy i rasy postaci, zmieniające częściowo rozgrywkę. Tak jak przy „MiniDungeons" i tutaj skupiono się głównie na algorytmach ewolucyjnych. Swego czasu popularną grą do tworzenia botów był „NetHack". Jest to jednak bardziej skomplikowana gra od wymienionych wcześniej, w której zaprogramowany był udźwig postaci i efekty czynów zależne od tego, jaką klasę się wybrało.

1.4. CodinGame

1.4.1. Opis platformy

CodinGame[3] jest platformą służącą do nauki programowania i rozwoju umiejętności programistycznych. Skupia się wokół zawodów, podczas których uczestnicy konkurują w pojedynkach botów. Na platformie dostępne są trzy typy gier:

- multiplayer,
- puzzle,
- gra optymalizacyjna.

W grze multiplayer bot walczy na planszy przeciwko kilku botom innych graczy. W grze typu puzzle zadaniem bota jest przejść przygotowane testy. Gra optymalizacyjna to specjalny typ gry puzzle, w której głównym celem jest nie tylko przejście planszy, ale i zmaksymalizowanie otrzymanego wyniku. Gra „Roguelike" opisywana w niniejszej pracy jest grą optymalizacyjną.

1.4.2. Framework

CodinGame udostępnia na Githubie zestaw narzędzi[4] oraz szkielet gry[5]. W skład narzędzi wchodzi `GameManager` i `GraphicEntityModule`. Napisane są one w Javie, a do wyświetlania widoku gry wykorzystują Javascript z biblioteką do renderowania grafiki PixiJS. Do zadań `GameManagera` należy wywoływanie kolejnych rund gry, wysyłanie danych do innych komponentów i ustawienia gry (takie jak maksymalna liczba tur i ich czas). `GraphicEntityModule` pozwala na rysowanie prostych

elementów planszy. Dostępna jest też klasa `GameRunner`, pozwalająca na lokalne testowanie programu.

Główną częścią, za którą odpowiada developer, jest klasa `Referee`. Klasa ta jest łącznikiem pomiędzy `GameManager` a klasą `Player`, która odpowiada za wywoływanie bota. `Referee` musi implementować trzy metody: `init()`, `gameTurn()` oraz `onEnd()`.

W `init()` należy pobrać z `GameManager` planszę testową i przygotować wszystkie dane potrzebne do gry – w przypadku gry „Roguelike" jest to stworzenie planszy oraz umieszczenie na niej gracza, potworów i skarbów. Metoda `gameTurn()` wysyła dane do klasy `Player` (jako `String`), pobiera od niej listę rozkazów bota i na ich podstawie aktualizuje stan gry. W przypadku „Roguelike" wysłana zostaje aktualna lista potworów i skarbów na planszy, a pobierana lista poleceń bota może zawierać rozkazy przejścia na inne pole lub zaatakowania przeciwnika. Metoda `onEnd()` określa, co ma się wydarzyć w przypadku przegranej i wygranej, takie jak wyświetlenie komunikatu czy zmiana punktacji. Innymi elementami, za które odpowiada developer, są plansze testowe oraz przykładowe boty.

Rozdział 2.

Implementacja

2.1. Wymagania

Założenia gry, jako roguelike'a, są następujące:

- bot porusza się po labiryncie turowo,
- bot ma określoną liczbę punktów wytrzymałości, które traci w czasie potyczek z przeciwnikami,
- bot ma podstawowy, darmowy atak, którym może zaatakować przeciwnika na sąsiednim polu oraz punkty magii, za które może wykonywać ataki o większym zasięgu lub większych obrażeniach,
- na planszy znajdują się mikstury: dodające punkty wytrzymałości lub magii, zwiększające zadawane obrażenia lub wynik końcowy,
- bot przegrywa, jeśli wynik końcowy lub liczba punktów wytrzymałości spadną do zera, lub przekroczony zostanie limit czasowy.

Dodatkowym celem projektu, którego ta praca jest częścią, jest opublikowanie gry na CodinGame. Można to osiągnąć na dwa sposoby. Pierwszy z nich to normalne umieszczenie gry, która wymaga zatwierdzenia przez użytkowników będących na odpowiednim poziomie zaawansowania. Drugą opcją jest opublikowanie gry w postaci oficjalnych zawodów, co jest celem przygotowywanej pracy. Ponieważ w takim przypadku CodinGame bierze udział w przygotowywaniu konkursu, stawiane są przed projektem dodatkowe wymagania:

- powinno być możliwe przedstawienie stanu planszy za pomocą prostych w obsłudze danych w ograniczonej liczbie linii (CodinGame generuje szkielet programu razem z danymi wejściowymi dla kilkunastu języków programowania),

- napisanie prostego bota powinno być kwestią kilkunastu linii kodu (początkująca osoba powinna być w stanie podjąć się gry),
- powinno być możliwe przejście gry w skończonej liczbie tur (około 200) ,
- przygotowanie przez bota odpowiedzi nie powinno zajmować dużo czasu (około 50ms, z ewentualnym dłuższym czasem na pierwszą turę),
- preferowana jest gra z pełną informacją (wynik bota nie powinien zależeć w głównej mierze od losowych czynników),
- dla gry optymalizacyjnej przyjęte jest, że całe wyjście programu można przedstawić już w pierwszej turze gry.

Powyższe wymagania mocno wpłynęły na ostateczny kształt gry. Początkowo przeciwnicy na planszy mogli się poruszać, gracz dysponował atakami obszarowymi, a całe lochy przysłaniała mgła wojny. Po konsultacji z CodinGame elementy te zostały odrzucone, ponieważ zbyt komplikowały rozgrywkę i wprowadzały za duży element losowości, który mógłby w przyszłości przeszkodzić w sprawiedliwym ocenianiu wkładu graczy.

2.2. Opis programu

2.2.1. Zasady gry

Rysunek 2.1 przedstawia planszę do gry. Panel po lewej stronie zawiera:

- pole z opcjonalnym komunikatem od bota,
- aktualną liczbę punktów zwycięstwa, wytrzymałości i magii,
- opisy ataków w formie: liczba zadawanych obrażeń / koszt w punktach magii.

Typy ataków podane są w tablicy 2.1. Zasięg podany jest jako odległość Manhattan.

Bot w swojej turze może się przesunąć lub zaatakować przeciwnika. Następnie rozpatrywana jest tura przeciwników. Jeśli po jej zakończeniu bohater żyje i znajduje się na polu z miksturą, automatycznie ją zbiera.

Po każdej turze bot traci określoną liczbę punktów zwycięstwa. Jeśli spadną one do zera, gra zakończy się porażką.

2.2.2. Struktura programu

Program można podzielić na cztery części:

Rysunek 2.1: Plansza gry.

Broń	Id	Zasięg	Koszt	Obrażenia
	0	1.0	0	5
	1	1.5	1	3
	2	2.9	2	5
	3	3.0	4	3

Tablica 2.1: Typy ataków gracza.

- config – informacje dla CodinGame,
- inputs – przypadki testowe w formie plików tekstowych, z których generowane są Jsony,
- src/main – Referee parsujące polecenia od bota oraz silnik gry,
- src/test – GameRunner i boty.

Potwór	Id	Zasięg	Wytrzymałość	Obrażenia
	1	1.0	15	3
	2	1.5	20	6
	3	2.9	15	3
	4	2.0	15	2
	5	2.0	30	7
	6	2.5	45	10

Tablica 2.2: Rodzaje przeciwników.

2.2.3. Konfiguracja, dane wejściowe i wyjściowe

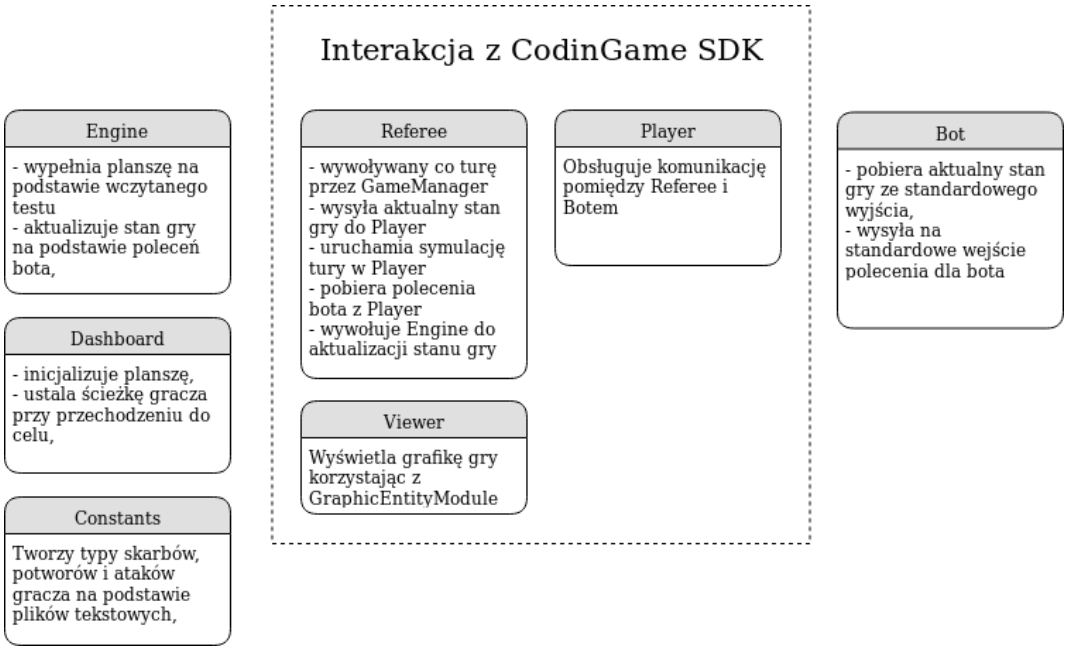
Config zawiera dodatkowe informacje potrzebne do umieszczenia gry na CodinGame. Wśród nich jest m.in. `Stub`, w którym w pseudokodzie zapisane są dane wejściowe dla programu. Służy on do generacji pliku początkowego w każdym z kilkunastu dostępnych na CodinGame języku programowania. Drugim plikiem potrzebnym platformie jest `statement_en`, który dokładniej opisuje jakie polecenia przyjmuje bot, jaki jest cel gry, ile czasu ma gracz na przedstawienie wyniku oraz jakie są warunki zwycięstwa i porażki. W config znajdują się też przypadki testowe, parsowane z plików tekstowych z folderu inputs do formatu Json za pomocą biblioteki Gson.

2.2.4. Silnik gry

Główna funkcjonalność programu znajduje się w pakiecie `engine`. Klasa `Engine` tworzy planszę i wszystkie obiekty na podstawie planszy wczytanej z pliku Json. W czasie gry `Engine` otrzymuje od Referee pojedyncze rozkazy i aktualizuje stan gracza na podstawie rozkazu. Po zaktualizowaniu gracza `Engine` uaktualnia stan potworów, skarbów i punktację. Przyjęte jest, że `Engine` otrzymuje poprawnie zbudowane polecenia, ale musi sprawdzić, czy są one możliwe do wykonania (np. czy polecenia ruchu nie zakończy się poza planszą).

Zadaniem Referee jest pobranie rozkazów od bota, sprawdzenie, czy rozkazy są poprawnie zbudowane, przekazanie ich do klasy `Engine` i zakończenie gry, jeśli wystąpiły warunki zwycięstwa lub porażki. Jeśli gra się nie zakończyła, Referee wysyła do bota aktualny stan planszy.

Statystyki wszystkich obiektów na planszy przechowywane są w plikach tekstowych w folderze resource. Klasa `Constants` pobiera je przy starcie programu i tworzy listy typów potworów i mikstur. Klasa `Engine` przy tworzeniu planszy z pliku Json pobiera id typu oraz jego współrzędne i tworzy w danym miejscu obiekt (o ile jest to możliwe).



Rysunek 2.2: Układ programu. Referee, Player i Viewer bezpośrednio komunikują się z CodinGame SDK. Engine, Dashboard i Constants odpowiadają za stworzenie i aktualizowanie świata gry.

2.3. Opis botów

2.3.1. Rozwój programu

W rozdziale o wymaganiach CodinGame wspomniane było, że opisywana gra tworzona była w kilku iteracjach. Tworząc projekt, trzeba do prototypowej wersji całego programu dostarczyć także boty, aby łatwiej było ocenić poziom skomplikowania rozgrywki, jak i sam pomysł. Spowodowało to, że wraz z kolejnymi iteracjami programu koncepcja botów się zmieniała. W oryginalnej wersji, z przeciwnikami o dużym zasięgu wzroku mogącymi znaleźć drogę do gracza, boty skupiały się bardziej na manewrowaniu i odciąganiu przeciwników od skarbów niż na walce. Kolejna wersja programu pociągnęła za sobą uproszczenie wrogów. Z powodu zmniejszonego pola widzenia taktyka odciągania wrogów stała się mało atrakcyjna, ponieważ żeby w ogóle zostać zauważonym trzeba było zbliżyć się na małą odległość, niepozwalającą na łatwy odwrót. W tej wersji najbardziej liczyło się ustawianie się w odpowiednim

miejscu i atakowanie przeciwników z daleka, dopóki walka wręcz stawała się nieunikniona. W wersji ostatecznej potwory nie poruszają się. Oznacza to, że na odciąganie przeciwników od kluczowych pól nie ma już miejsca i ważniejsze stało się szacowanie przebiegu całej rozgrywki niż reagowanie zależnie od sytuacji.

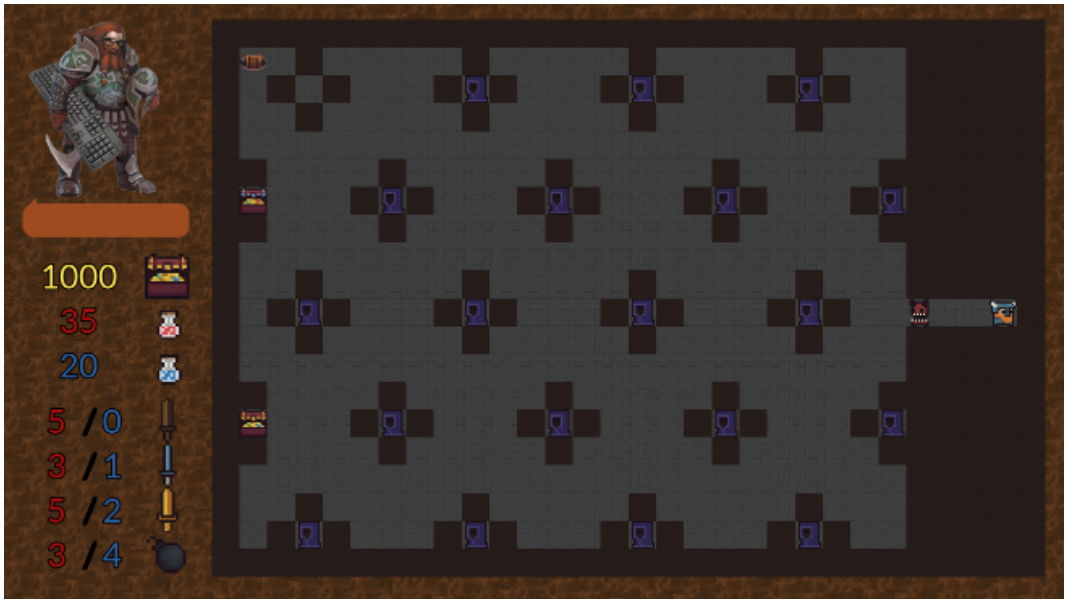
2.3.2. Boty

W obecnym momencie dostępne są dwa boty. Pierwszy przeprowadza prostą analizę kilku możliwych ścieżek, dla każdej sprawdza możliwość przeżycia i wybiera najlepszą opcję. Drugi bazuje na przechowywaniu stanów gry dla możliwych ścieżek i szukaniu kolejnych rozwiązań dla najlepszego z nich.

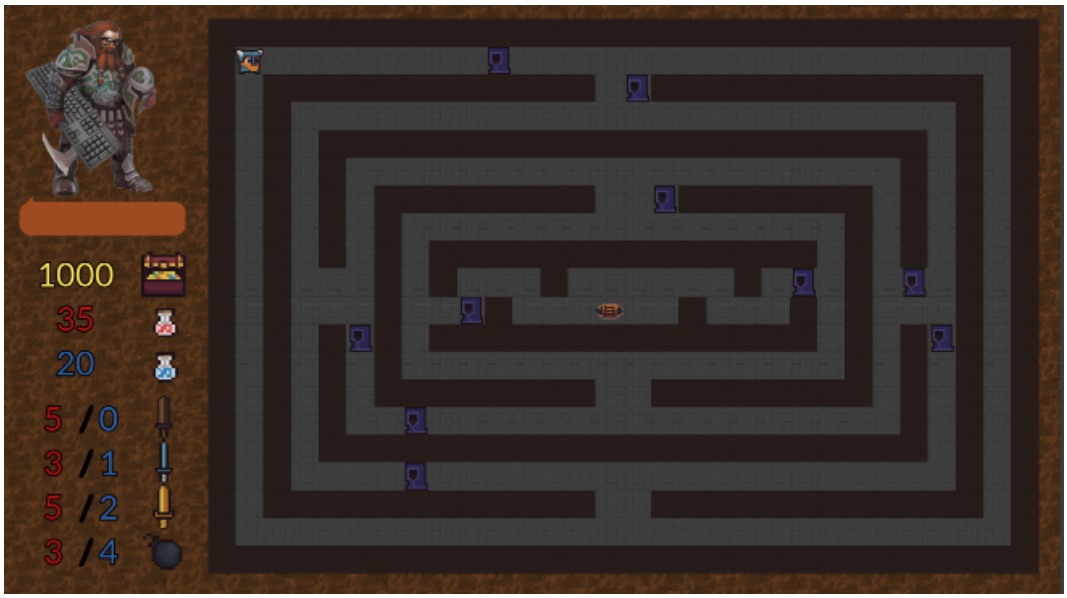
Pierwszy bot układa najkrótsze ścieżki do każdego z wyjść i kalkuluje, czy gracz jest w stanie przeżyć drogę. Jeśli nie znajdzie żadnej takiej ścieżki, próbuje znaleźć dłuższą, ale bezpieczniejszą i sprawdza, czy po drodze nie jest w stanie zebrać mikstur, które zwiększą szanse na przetrwanie. W tym celu liczone są ścieżki dla trzech różnych sytuacji: ignorującej potwory na planszy, omijającej wszystkie potwory na planszy oraz omijającej wybrane potwory na planszy (do unikania ataków potworów po drodze i do atakowania potwora z miejsca, w którym gracz nie jest narażony na więcej obrażeń).

Drugi bot lokalnie rozstrzyga sytuację podobnie do pierwszego, ale działa według innego schematu. Przeprowadza symulację ruchów do kilku najbliższych obiektów na planszy, oblicza statystyki gracza i stan gry (współrzędne gracza, pozostałe obiekty na planszy, punkty zwycięstwa, wykonane jak dotąd rozkazy, głębokość symulacji). Następnie dla każdego z obiektów, do których gracz dotarł, rekurencyjnie wywoływana jest kolejna symulacja. Na określonej głębokości przeszukiwania program zwraca ostateczny stan gry, spośród których wybierany jest ten, w którym gracz zachował jak najwięcej punktów wytrzymałości przy jak największej liczbie punktów zwycięstwa. Po znalezieniu najlepszej wersji bot zostaje po wykonaniu rozkazów od razu skierowany do wyjścia.

O ile oba boty są w stanie przejść proste plansze, to drugi algorytm nie radzi sobie z tendencyjnie stworzonymi planszami, tj. takimi, gdzie ważne jest rozeznanie się w całej planszy, ponieważ składa on ścieżkę krok po kroku, nie biorąc pod uwagę szerszego obrazu.



Rysunek 2.3: Normalna plansza – manewrowanie między przeciwnikami.



Rysunek 2.4: Plansza, na której lokalne rozwiązywanie problemu nie zadziała. Drugi bot będzie próbował pokonać po kolei każdego spotkanego wroga, jednak pokonanie każdego na swojej drodze jest niemożliwe, ponieważ zabraknie mu zarówno punktów wytrzymałości jak i punktów magii na ostatniego przeciwnika. Pierwszy bot zaatakuje tylko pierwszego i ostatniego wroga na swojej drodze, a obok reszty po prostu przejdzie, tracąc mniej punktów wytrzymałości niż gdyby miał stanąć do walki z każdym z nich.

Rozdział 3.

Tworzenie plansz i bota

3.1. Tworzenie plansz

Plansze dla gry przechowywane są w następującej formie:

- 20 linii po 30 znaków – układ planszy, gdzie `.` oznacza puste pole, a `#` – ścianę,
- n linii oznaczających obiekty na planszy, zapisane jako trójki trzech liczb: id obiektu – współrzędna x – współrzędna y.

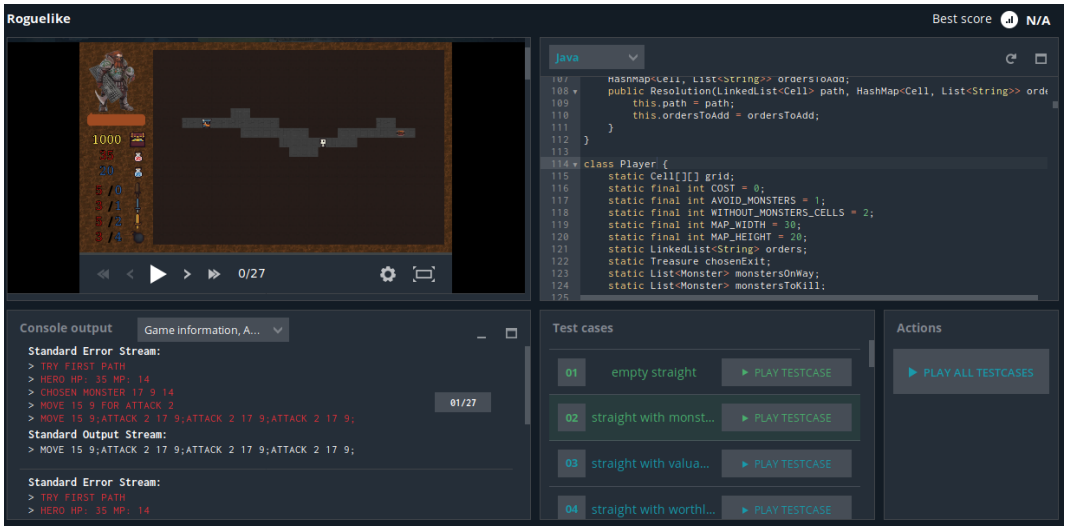
W klasie `Main`, przed uruchomieniem bota, wszystkie plansze zostają zapisane w formacie Json, z dodatkowymi informacjami, których CodinGame potrzebuje do uruchomienia planszy. Przy uruchomieniu kodu, klasa `Engine` wyłuskuje z testu opis planszy i na jej podstawie tworzy ściany i obiekty.

3.2. Tworzenie własnego bota

Tworzenie bota można realizować na dwa sposoby: albo poprzez edytor dostępny na CodinGame, albo poprzez stworzenie lokalnie klasy Bota. Bot musi zostać uruchomiony w klasie `Main` razem z nazwą testowanej planszy.

3.3. Wejście i wyjście programu

Po wybraniu na CodinGame języka programowania, w którym chcemy pisać bota, wyświetlany jest szkielet programu zawierający przykład pobierania danych wejściowych i wypisywania danych wyjściowych dla danej gry. Aby możliwe było wygenerowanie takich danych, do każdej gry należy dołączyć plik `Stub`. Znajdują się w nim dane wejściowe napisane w przygotowanej przez CodinGame składni.



Rysunek 3.1: CodinGame online IDE z wbudowanym edytorem kodu i narzędziami do uruchamiania przykładowych testów.

Ponieważ program działa w niekończącej się pętli, wejście programu dzieli się na dane przysyłane tylko w pierwszej rundzie (np. plan labiryntu) jak i na te otrzymywane co turę (jak opis potworów):

```
loop 20 read row:string(30)
read score:int

gameloop
read heroHp:int heroMp:int heroX:int heroY:int
loop 4 read attackId:int attackDmg:int
read entitiesCount:int
loop entitiesCount read entityId:int entityType:int entityX:int entityY:int entityValue:i
write join("MOVE 1 1")
```

Po przeprowadzeniu obliczeń bot powinien zwrócić jako wynik ciąg zdań rozdzielonych średnikami o możliwej treści:

- `MOVE X Y napis` – przesuń bota na współrzędne X i Y. Program przesuwa bota po najkrótszej możliwej drodze (wybierając po kolei kierunki: północ, wschód, południe, zachód). W przypadku natrafienia na potwora zostaje automatycznie wykonywany darmowy atak,
- `ATTACK ID X Y napis` – zaatakuj potwora na współrzędnych X i Y atakiem o podanym ID.

W polu `napis` może być podany dowolny tekst – wyświetli się on w panelu gracza w czasie jego tury. Rozkazy można wysyłać do Referee pojedynczo lub wiele na raz.

3.4. Warunki zwycięstwa i porażki

Warunkiem zwycięstwa jest dotarcie do mety w wyznaczonej liczbie tur z wynikiem większym od zera. Warunki porażki to:

- spadek liczby punktów wytrzymałości do zera,
- przekroczenie liczby tur,
- dotarcie do mety z liczbą punktów zwycięstwa mniejszą lub równą 0. Dodatkowe informacje o oczekiwanym działaniu bota znajdują się w pliku `statement_en`.

Rozdział 4.

Uruchomienie programu

4.1. Lokalnie

Aby uruchomić program lokalnie potrzebne są:

- maven (do zbudowania programu),
- przeglądarka wspierające Javascript wersję ES6 (do lokalnego testowania),
- Java IDE takie jak IntelliJ lub Eclipse (opcjonalne, zalecane przez CodinGame).

4.2. Przez stronę CodinGame

Grę można uruchomić też poprzez stronę CodinGame[7]. Pozwala to na pisanie bota w różnych językach, wyświetlanie wyników testów i ich masowe uruchamianie.

Rozdział 5.

Podsumowanie

W pracy omówione zostały dwie kwestie: stworzenie samego silnika gry oraz botów przechodzących grę. Wyszczególniono również warunki, jakie musi spełniać cały program dla przyszłej akceptacji na platformie CodinGame. Boty były testowane na kilkunastu przygotowanych planszach, w celu sprawdzenia poprawności programu i przekonania się, czy gra nie ma uchybień, które umożliwiałyby łatwe znalezienie optymalnego rozwiązania.

Praca nad projektem była utrudniona z powodu wymagań stawianych przez CodinGame, z których części udało się sprostać przy okazji kolejnych iteracji pisania silnika gry. W porównaniu do innych gier optymalizacyjnych na CodinGame łatwo jest stworzyć planszę, która, zamiast ograniczyć wynik prostego bota (kilkanaście linii dla początkującej osoby), pokona go w pierwszych kilkunastu turach. Takich problemów nie mają takie gry, jak np. „AStarCraft", w której wynik końcowy nie zmieni się diametralnie przez jedno uchybienie. Dodatkowo napisanie algorytmu, który będzie umiał więcej niż tylko przemieszczać się między wybranymi punktami, wymaga poświęcenia większej liczby czasu. Powoduje to, że cały projekt prawdopodobnie będzie mieć wyższy próg wejścia niż przeciętna gra na CodinGame. Można temu częściowo zaradzić poprzez tworzenie mniejszej liczby plansz z przypadkami brzegowymi, które były wykorzystane przy testowaniu przykładowych botów.

Mając przygotowaną część projektu zaprezentowaną w powyższej pracy możliwe jest dalsze poprawianie grywalności i w niedługiej przyszłości, przedstawienie CodinGame ulepszonej wersji gry i umieszczenie jej na platformie w postaci oficjalnych zawodów.

Bibliografia

[1] Christoffer Holmgard, Antonios Liapis, Julian Togelius and Georgios N. Yannakakis, *Evolving Personas for Player Decision Modeling*, 2014

[2] Vojtech Cerny, Filip Dechterenko. *Rogue-Like Games as a Playground for Artificial Intelligence – Evolutionary Approach.* 14th International Conference on Entertainment Computing (ICEC), Sep2015, Trondheim, Norway. pp.261-271

[3] CodinGame, `https://www.codingame.com/`

[4] silnik CodinGame, `https://github.com/CodinGame/codingame-game-engine` (na dzień 30.08.2019)

[5] szkielet gry CodinGame, `https://github.com/CodinGame/game-skeleton` (na dzień 30.08.2019)

[6] BotHack – A Nethack Bot Framework, `https://github.com/krajj7/BotHack` (na dzień 30.08.2019)

[7] Roguelike, `https://www.codingame.com/ide/demo/808594edbcae72480c07b988218913c5b3dec9` (na dzień 30.08.2019)