Projekt - Azul

-

APRO2 2023L

Etap I - Aleksander Belazerski, Stanisław Ciszkiewicz, Stanisław Dolinski, Michał Kozłowski, Jan Sakowski
Etap II - Maja Berej, Mateusz Orzełowski, Krzysztof Czaplicki, Rafał Kowalczuk

Politechnika Warszawska, Cyberbezpieczeństwo

20 maja 2023

Spis treści

1. Założenia Projektu

Projekt z przedmiotu APRO2 składa się z trzech etapów:
— Etap I - design proposal (projekt aplikacji, wykresy UML)
— Etap II - backend (logika palikacji, testy)
— Etap III - frontend (interfejs aplikacji, testy)
Zadanie projektowe naszej grupy zostanie pfrzekazane po pierwszym etapie innemu zespołowi, który przperowadzi analize naszej pracy i zrealizuje etap II. Sprawozdanie to będzie doukumentacją projektową zadania do końca semestru i zostanie przekazane kolejnym grupom realizującym następne etapy.

2. Opis gry - Azul

Azul jest grą logiczną dla od 2 do 4 osób. Każdy gracz otrzymuje na stracie rozgrywki planszę, w której skład wchodzą: ściana (wall), podłoga (floor),linie wzoru (pattern lines) oraz tabelkę z punktacją. Na stole stawiamy 2n+1 warsztatów (gdzie n to liczba graczy), na które losujemy po 4 kafelki z worka (bag). Celem rozgrywki jest zdobycie jak największej ilości punktów poprzez ułożenie mozaiki na ścianie (dokładna punktacja zawarta jest w oficjalnej instrukcji gry). W skład gry wchodzi 5 rodzajów kolorowych kafelków (5x20) oraz kafelek pierwszego gracza, który na początku każdej rundy umieszczany jest na środku. W czasie jednej rundy gracz wybiera warsztat (factory) i zabiera z niego wszystkie kafelki wybranego przez niego koloru - reszta kafelków zrzucana jest na środek stołu (center). W późniejszych etapach gry gracz może dobrać wszystkie kafelki wybranego koloru ze środka stołu (pierwsza osoba, która wykona takowy ruch zabiera również kafelek pierwszego gracza i umieszcza go na swojej podłodze). Zabrane kafelki ustawia na liniach wzorów, tak aby w jednej linii znajdowały się tylko i wyłącznie kafelki jednego koloru. Kafelki, których nie jesteśmy w stanie ułożyć na liniach wzorów zrzucane są na podłogę. Po zakończeniu rundy (wyczerpaniu zasobów kafelków na stole) gracze z zapełnioną linią wzoru przekłada jeden z kafelków na ścianę (resztę odrzuca do pudełka(box)). W tym samym momencie gracze zliczają punkty ujemne utracone za rundę (kafelki z podłogi również odkładamy do boxa). Następnie rozpoczynana jest kolejna runda gry - w razie braku kafelków w worku, uzupełniany jest on zawratością pudełka. Rozgrywka toczona jest do momentu, w którym któykolwiek z graczy zapełni cały wiersz w ścianie.

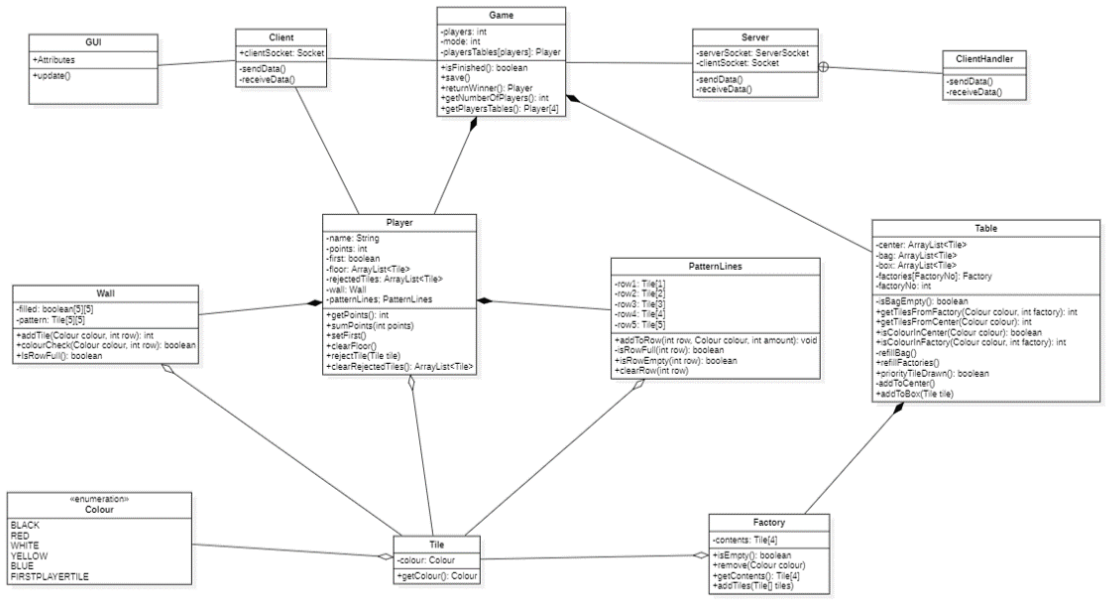
Oficjalna instrukcja gry znajduje się na stronie: https://www.ultraboardgames.com/azul/game-rules.php

3. Etap I

W tym etapie zaprponujemy realizację zadania projektowego opartą na projekcie klas (diagramy UML) oraz założeniach naszej aplikacji.

3.1. Podział na klasy

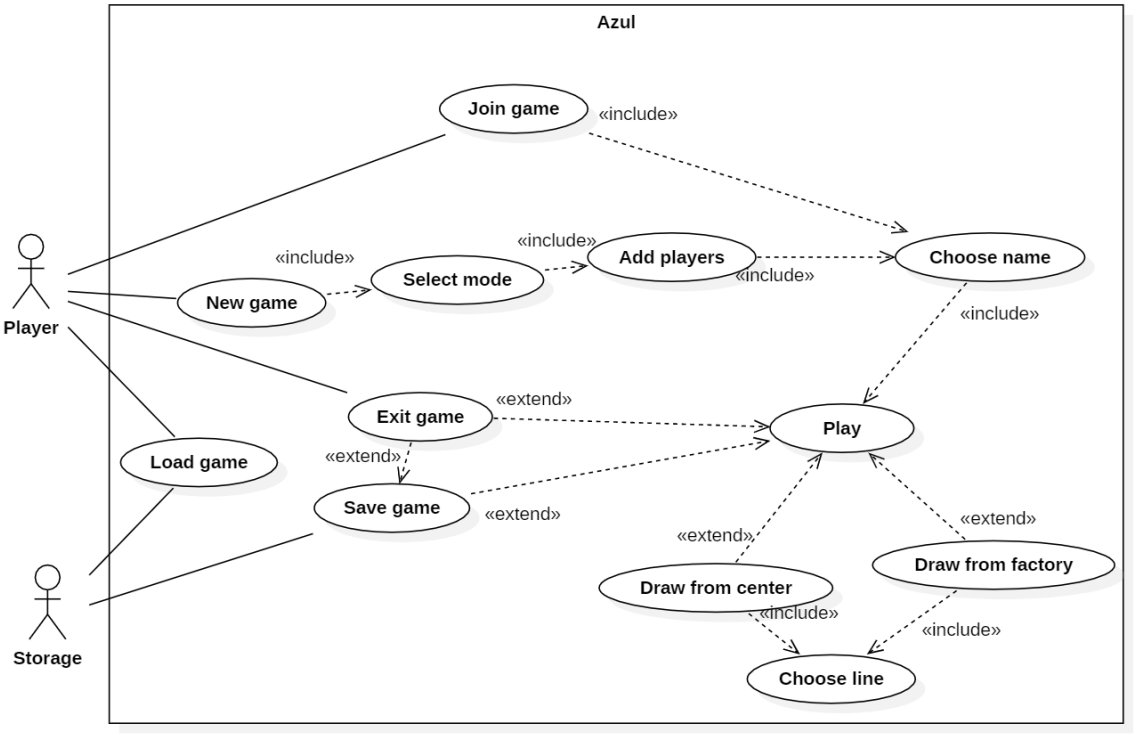
Nasza aplikacja docelowo powinna składać się z następujących klas:
— klasa Player - reprezentuje gracza oraz jego planszę, przechowuje informacje o graczu (imię, aktualne punkty itd.)
— klasa Wall - reprezentuje kolorową ścianę, na której gracz układa swoje kafelki
— klasa PatternLines - reprezentuje linie wzorów na której gracz przygotowywuje kafelki do ułożenia na ścianie
— klasa Tile - reprezentuje jeden z 6 kafelków (5 kolorowych i 1 kafelek pierwszego gracza)
— klasa Table - reprezentuje stół, na którym odbywa się rozgrywka
— klasa Factory - reprezentuje warsztat, z którego garcze dobierają kafelki
— klasa Game - odpowiada za komuniukację między klasą Table a Player, posiada informacje o stanie rozgrywki
— klasa Server - odpowiada za możliwość rozgrywki online
— klasa ClientHandler - odpowiada za komunikację Client-Server
— klasa Client - klient odpowiada za gracza w przypadku gry online, a w przypadku gry offline przjmuje funkcje serwera
— klasa GUI - interfejs graficzny użytkownika



Szczegółowy opisa klas (pól i metod) zawarty jest w dokumencie javadoc dostępnym w repozytorium projektowym.

3.2. Use-case diagram

Przygotowaliśmy również diagram przypadków użycia danych klas naszego programu:



4. Etap II

Ten etap projektu obejmował implementację kodu programu na podstawie diagramu UML oraz *use-case diagram* w języku *Java.*

4.1. Zmiany

4.1.1. Klasa Colour i Tile

Podjęliśmy decyzję o integracji klasy Colour z klasą typu enum Tile z uwagi na to, że klasa Colour zawierała tylko jeden obiekt oraz metodę wywołującą ten obiekt.

4.1.2. Metody

Ze względu na wygodę oraz większą przejrzystość kodu zdecydowaliśmy się stworzyć znaczącą ilość nowych metod. Dokładne zastosowanie pierwotnych oraz nowych metod opisane jest w dokumencie *Javadoc.* Kilka z przykładowych dodanych przez nas nowych metod to:
— containsTile() - sprawdza, czy w danym wierszu znajduje się kafelek o podanym kolorze
— getName() - zwraca imię gracza
— isFirstStageFinished() - sprawdza, czy ukończone zostało wybieranie kafelków z warsztatów oraz środka stołu
— isMoveValid() - sprawdza, czy ruch może zostać wykonany
— addTilesToPatternLines() - dodaje wybrane kafelki do *pattern linesów.*

4.1.3. Diagram UML

W związku ze stworzeniem nowych metod oraz subtelnymi zmianami w strukturze klas względem pierwotnych założeń, diagram UML znacznie się zmieni. Jego obecna forma przedstawiona jest poniżej.

4.2. Testy

W ramach tego etapu projektu stworzyliśmy wiele testów, które okazały się bardzo pomocne w wykrywaniu błędów w programie. Do przeprowadzenia testów wykorzystaliśmy framework *JUnit*. Dodatkowo, w celu efektywniejszego odnajdywania błędów, stworzyliśmy wersję Azula dostępną w terminalu. Okazało się, że ta funkcjonalność również przyniosła znaczną pomoc w procesie znajdowania błędów w programie. Ta specjalna wersja programu pozwalała nam na przeprowadzanie testów w środowisku konsolowym, co było szczególnie przydatne przy debugowaniu i sprawdzaniu zachowania programu w różnych scenariuszach.

4.3. Analiza krytyczna

Uważamy, że zamysł przewidywanej struktury programu stworzony w etapie I był bardzo dobry biorąc pod uwagę złożoność programu. Zmiany, których dokonaliśmy - zintegrowanie dwóch klas oraz dodanie nowych metod - były zazwyczaj kwestią wygody. Dwie dodane przez nas metody były konieczne. Było to *isMoveValid()* i *addTilesToPatternLines()*. Bez nich nie można by było spełnić założeń gry. Jedną z rzeczy, która w znacznym stopniu utrudniła nam pisanie programu pod względem technicznym było ubogie przedstawienie aspektów sieciowych.

4.4. Wnioski

Podsumowując, etap II był bardzo pouczający. Po raz pierwszy implementowaliśmy program na podstawie diagramu UML. Zadanie to okazało się niezywkle wymagające, głównie ze względu na to, że nie jest łatwo zrozumieć wizję autorów diagramu UML. Nawet drobna różnica w interpretacji metody może skutkować znaczącą zmianą struktury programu, m. in. przez co nasz diagram UML różni się od pierwotnego. Brak kontroli nad projektem od samego jego początku to nowe dla nas przeżycie, przez które musieliśmy nauczyć się dostosowywać do narzuconych z góry rozwiązań. Ze względu na świadomość, że nasz kod zostanie przekazany kolejnej grupie, staraliśmy się zachować jak największą przejrzystość stworzonych metod oraz rozbudowywać dokumentację tak, aby była ona zrozumiała i rzetelna.