

Projekt - Azul

-

APRO2 2023L

Etap I - Aleksander Belazerski, Stanisław Ciszewicz, Stanisław Dolinski, Michał Kozłowski,
Jan Sakowski

Etap II - Maja Berej, Mateusz Orzełowski, Krzysztof Czaplicki, Rafał Kowalczyk

Etap III - Jakub Szweda, Mateusz Plichta, Karol Żelazowski, Maja Dąbrowska,
Maksymilian Młodnicki

Politechnika Warszawska, Cyberbezpieczeństwo

16 czerwca 2023

Spis treści

1. Założenia Projektu	2
2. Opis gry - Azul	2
3. Etap I	3
3.1. Podział na klasy	3
3.2. Use-case diagram	4
4. Etap II	4
4.1. Zmiany	4
4.1.1. Klasa Colour i Tile	4
4.1.2. Metody	4
4.1.3. Diagram UML	4
4.2. Testy	5
4.3. Analiza krytyczna	5
4.4. Wnioski	5
5. Etap III	5
5.1. Analiza etapu II	5
5.2. Tworzenie GUI	5
5.3. Funkcjonalność programu	6
5.4. Testy	6
5.5. Wygląd GUI	6
5.6. Film prezentujący działanie programu	8
5.7. Wnioski i przemyślenia	8

1. Założenia Projektu

Projekt z przedmiotu APRO2 składa się z trzech etapów:

- Etap I - design proposal (projekt aplikacji, wykresy UML)
- Etap II - backend (logika aplikacji, testy)
- Etap III - frontend (interfejs aplikacji, testy)

Zadanie projektowe naszej grupy zostanie przekazane po pierwszym etapie innemu zespołowi, który przeprowadzi analizę naszej pracy i zrealizuje etap II. Sprawozdanie to będzie dokumentacją 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 wchodzi: 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 przekładają 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 rozpoczyna się kolejna runda gry - w razie braku kafelków w worku, uzupełniany jest on zawartością pudełka. Rozgrzywka toczona jest do momentu, w którym którykolwiek z graczy zapełni cały wiersz w ścianie.

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

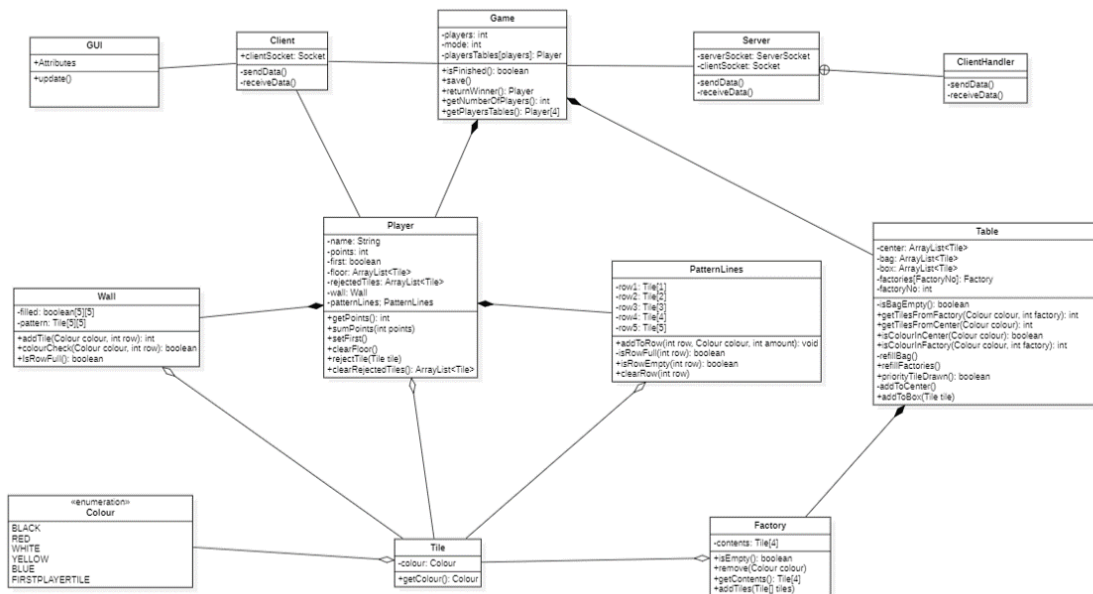
3. Etap I

W tym etapie zapraszamy do realizacji 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:

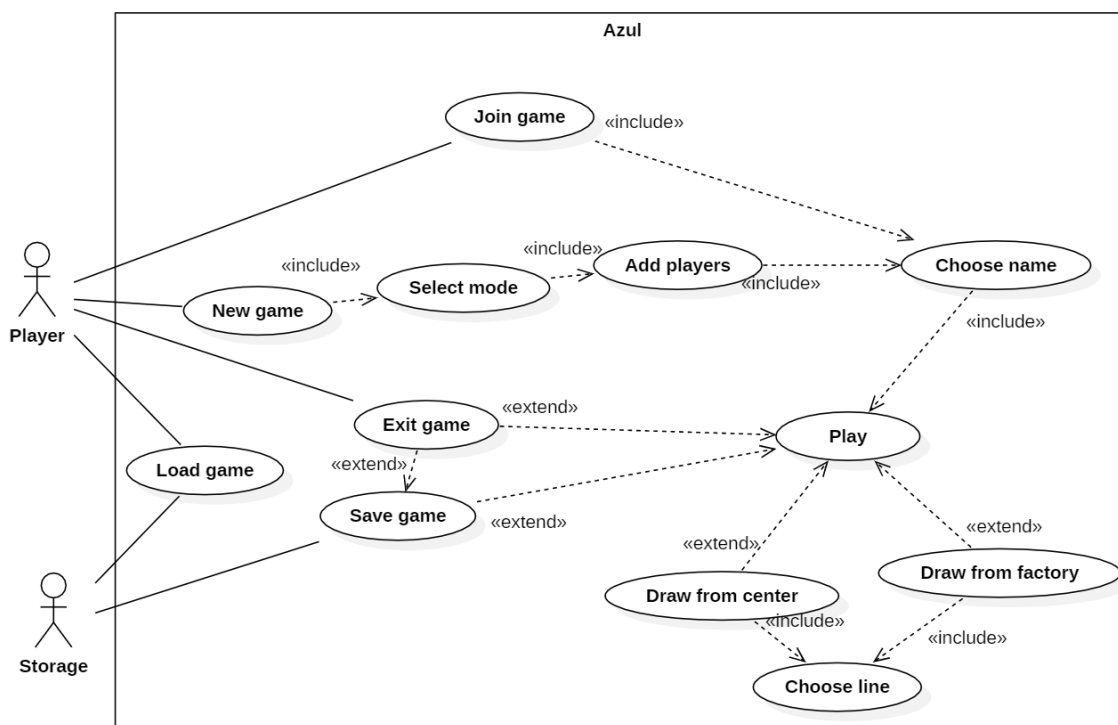
- klasa Player - reprezentuje gracza oraz jego planszę, przechowuje informacje o graczu (imię, aktualne punkty itd.)
- klasa Wall - reprezentuje kolorową ścianę, na której gracz układa swoje kafelki
- klasa PatternLines - reprezentuje linie wzorów na której gracz przygotowuje kafelki do ułożenia na ścianie
- 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 gracz dobiera kafelki
- klasa Game - odpowiada za komunikację 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 przymiemy funkcje serwera
- klasa GUI - interfejs graficzny użytkownika



Szczegółowy opis 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 niezwykle 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.

5. Etap III

Etap III Projektu obejmował implementację GUI programu korzystając z kodu zawartego w poprzednich etapach. Był to finalny etap projektu i wymagał on ukończenia wszystkich zadań i utworzenia działającego programu.

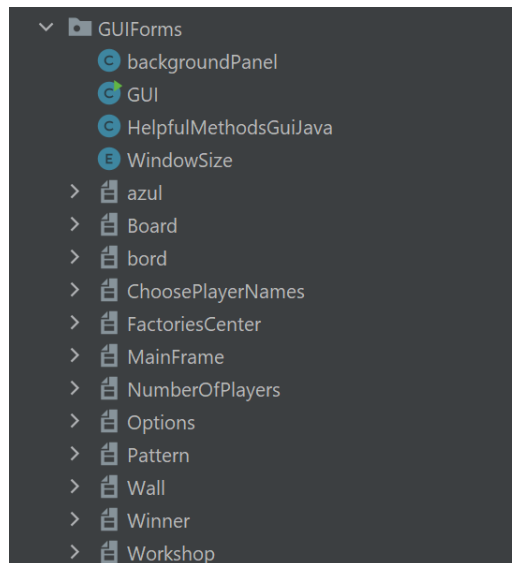
5.1. Analiza etapu II

Dzięki etapowi II otrzymaliśmy całą mechanikę gry i metody odpowiedzialne za prawidłowy przebieg gry. Bardzo pomocna była klasa *GUIGAME*, która odpowiadała za wszystko co powinno dziać się w GUI i być tam pokazywane.

Główną zmianą jaką wprowadziliśmy była edycja klasy *GUIGAME*, aby nie operowała ona na terminalu, lecz na zaprogramowanym przez nas GUI.

5.2. Tworzenie GUI

GUI zostało zaprogramowane korzystając z Java Swing w środowisku programistycznym IntelliJ. W tym celu utworzyliśmy wiele klas i powiązanych z nimi plików *.form* (co było wymagane przez wysoką złożoność działania gry). Do uruchomienia GUI służy klasa *GUI*, dodatkowo utworzyliśmy klasę *HelpfulMethodsGui*, która przydała się do tworzenia przycisków, etykiet i uzyskiwania obrazka o podanym rozmiarze ze ścieżki.



Rys. 1. Budowa GUI

5.3. Funkcjonalność programu

Nasz program pozwala na grę lokalną dla 2-4 graczy w grę planszową *Azul*. Dzięki wykorzystaniu wielu panelów gra jest przyjemna dla użytkownika i pozwala na operowanie na dużych planszach, co jest szczególnie przydatne na małych ekranach. Dodatkowo możliwe jest przełączanie pomiędzy planszą i warsztatami oraz warsztatami i centrum, co pozwala na podgląd każdego miejsca podczas tury gracza.

W celach rywalizacji, podczas gry wyświetlany jest stale panel z nazwami gracza oraz ich wynikami, co pozwala na śledzenie wyników. W celu upewnienia się kolejności tury, podczas tury obecnego gracza, jego nazwa jest podświetlana na zielono.

Dodatkowo w przypadku kiedy kończy się tura, podświetlone zostają płytki gracza, które zostały umieszczone na linii wzoru, co pozwala na potwierdzenie poprawności wyniku.

Kolejną przydatną funkcjonalnością jest możliwość zmiany rozdzielczości programu w sekcji *Options*, do wyboru mamy:

- 1280 x 720
- 1600 x 900
- 1904 x 1072

Dzięki tej opcji użytkownik może w łatwy sposób dostosować rozmiar gry na monitorze.

5.4. Testy

Podczas implementacji programu różne osoby testowały program, aby wyłapać ewentualne błędy. Dzięki temu znaleźliśmy dużo bugów z zakresu niepoprawnego dodawania płytek na planszę, braku aktualizacji wyników czy też zawieszania się programu. W ten sposób potrafiłmy naprawić wiele błędów oraz nanieść trafne blokady, takie jak niemożność naciśnięcia na kafelek na planszy, w którym już jest płytka.

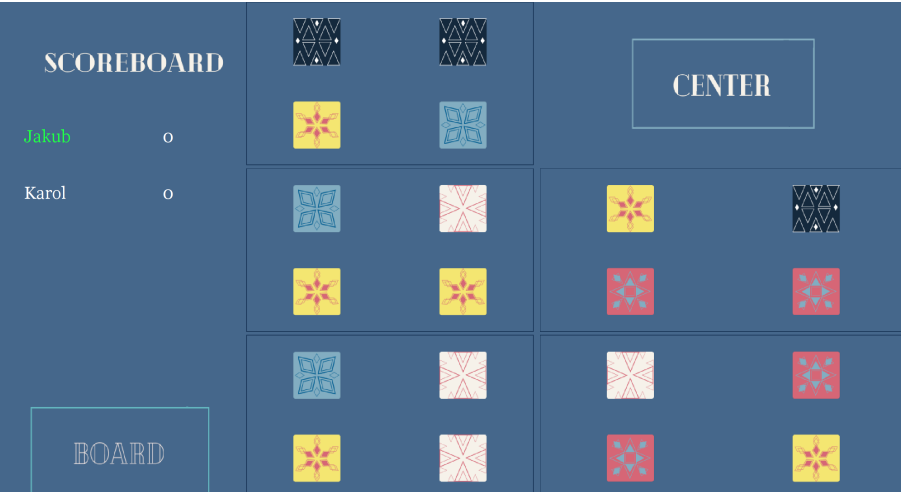
5.5. Wygląd GUI

Poniżej prezentujemy jak wygląda GUI do naszego programu, w kolejności:

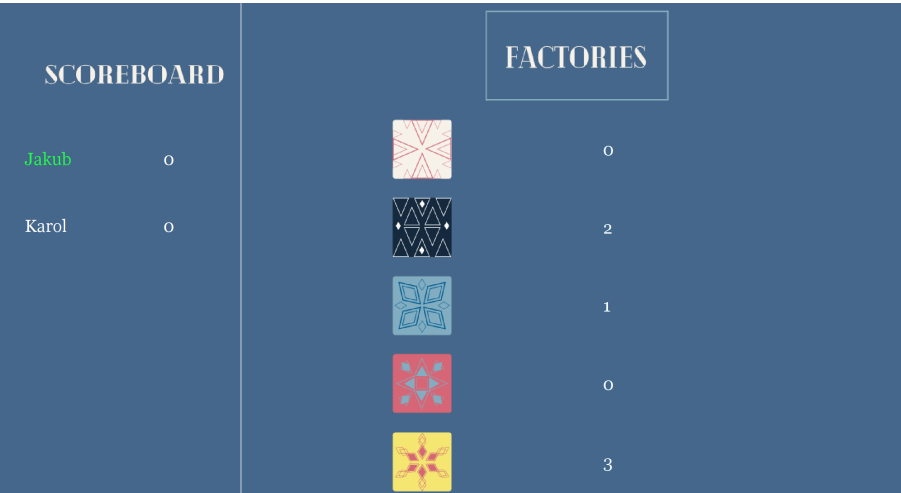
- Plansza wyboru liczby graczy
- Fabryki płytek
- Centrum
- Plansza gracza



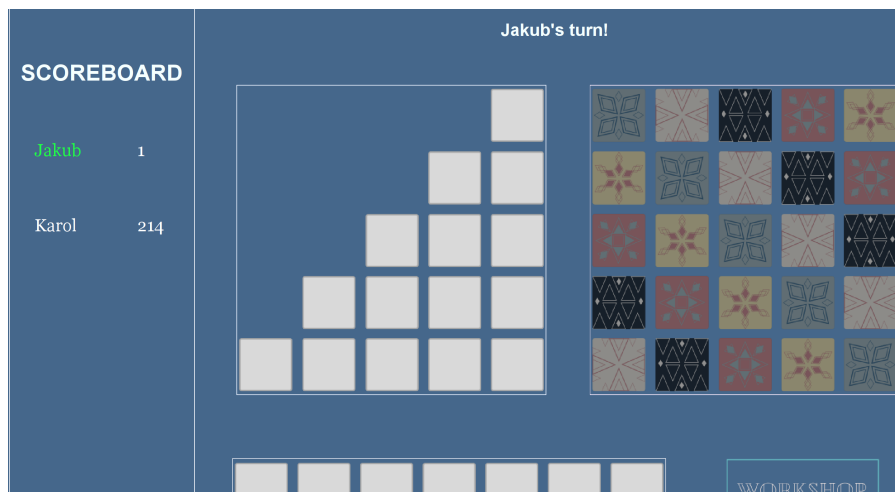
Rys. 2. Plansza wyboru liczby graczy



Rys. 3. Fabryki płytek



Rys. 4. Centrum



Rys. 5. Plansza gracza

5.6. Film prezentujący działanie programu

Poniżej zamieszczamy link do krótkiego filmu opisującego działanie naszego programu:

[Film Azul](#)

5.7. Wnioski i przemyślenia

Praca nad etapem III dała nam mnóstwo wiedzy na temat projektowania GUI w Java Swing oraz analizowania kodu napisanego przez inne grupy. Dodatkowo mieliśmy za zadanie przygotowywać grafiki do GUI, co również wymagało nowych umiejętności. Podsumowując, dzięki tworzeniu GUI oraz finalizowaniu programu, tak aby działał zgodnie z zasadami gry zdobyliśmy niezbędną wiedzę na temat projektowania podstawowych programów i aplikacji.