Politechnika Świętokrzyska w Kielcach

Wydział Elektroniki, Automatyki i Informatyki

Projekt **Programowanie w języku Java**

Temat: **Monopoly RPG**

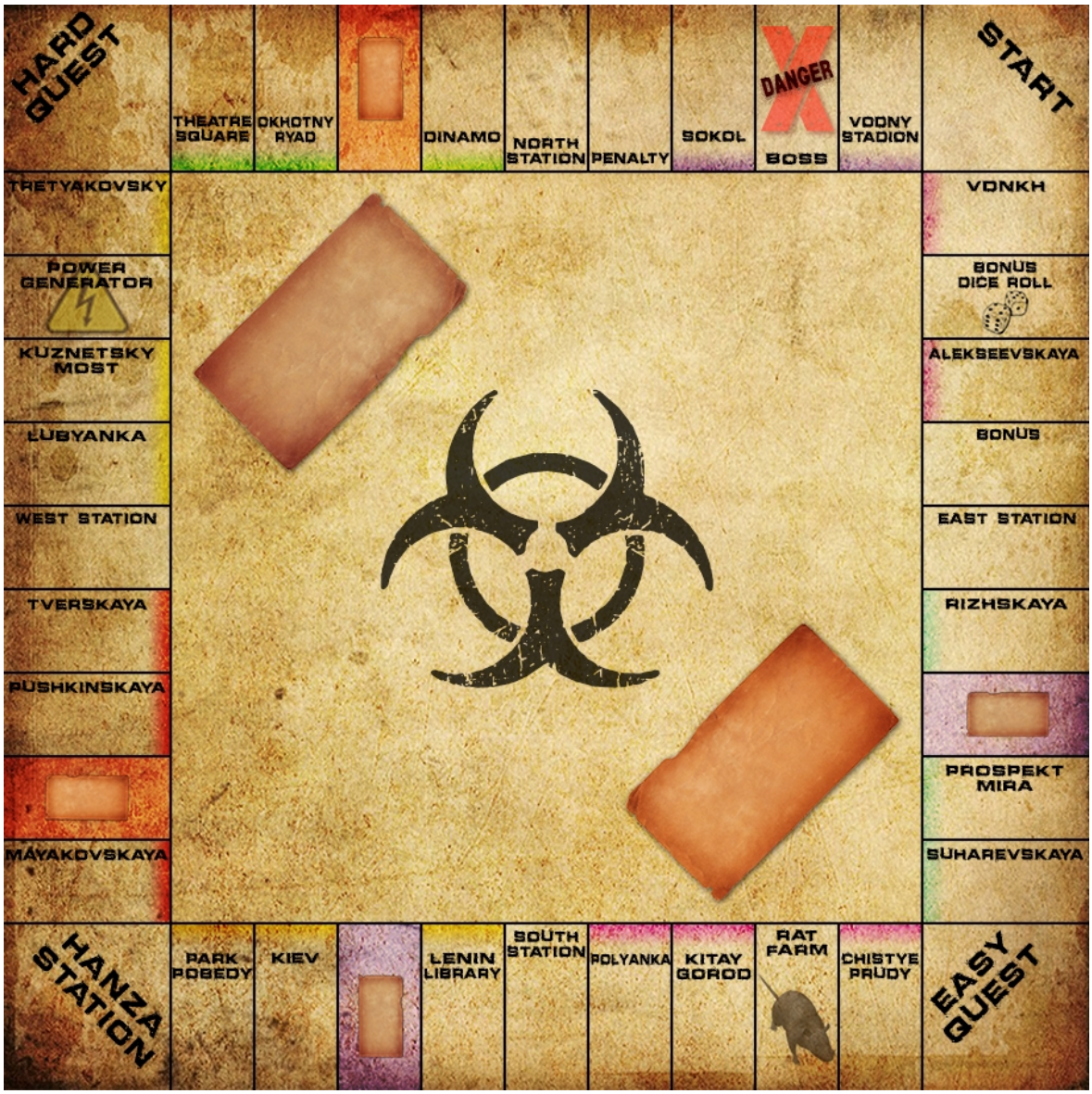
Studia **STACJONARNE I STOPNIA** Kierunek **INFORMATYKA** Grupa **214B**	Zespół: **1. Kacper Obara** **2. Emil Nowak**

1. Wstęp

Stworzony przez nas projekt pt. „Monopoly RPG” jest próbą połączenia dobrze znanej gry Monopoly z grą typu role-play, gdzie gracze, poza przejęciem planszy, mogą rozwijać własne postacie.

Wzorem do stworzenia klimatu było uniwersum Metro 2033 czyli świat postapokaliptyczny w którym umiejętność poradzenia sobie w zaistniałej sytuacji jest na wagę złota.

Niestety, nie udało nam się ukończyć właściwej gry w pełni. W obecnej postaci, jest w stanie grywalnego dema, natomiast do ukończenia w stu procentach gry, pozostało zrobić GUI, lepszą funkcjonalność gry oraz dodanie nowych przeciwników. Wiążemy z tym projektem nadzieję, dlatego planujemy dokończyć grę w okresie wakacyjnym.



Pole gry Monopoly RPG w klimacie postapokaliptycznym.

2. Zasady gry

Celem każdego z graczy jest stworzenie samowystarczalnej sieci stacji, zdolnej również do obrony przez innymi graczami jak i potworami. Również rozwijanie własnej postaci jest kluczem do zwycięstwa.

3. Sterowanie

- Przed rozpoczęciem gry:

Aby uruchomić rozgrywkę, jedna osoba musi stworzyć serwer i wybrać z menu opcję **„Host Game”**. Ta sama osoba jak i również pozostali gracze, muszą uruchomić ten sam program oraz dołączyć do gry poprzez wybranie z menu opcji **„Join Game”**.

Gdy połączy się 4 klientów, gra rozpocznie się automarycznie. W przeciwnym razie, jeśli graczy będzie mniej, osoba która tworzyła serwer musi nacisnąć spację aby rozpocząć grę.

- W trakcie gry:

SPACJA – rzut kośćmi (w przypadku jeśli trwa tura gracza).
Z – zakończenie obecnej tury gracza. Działa tylko po rzucie kośćmi.
Lewy CTRL, Lewy SHIFT – wyświetlenie informacji o statystykach postaci i jej ekwipunku.

Na odpowiednich polach, litery **Q** oraz **W** służą do akcji specjalnych. Powinny być one opisane w konsoli po stanięciu na pole.

Na przykład, na polu *SHOP*, litera Q służy do przeglądania zawartości sklepu. Natomiast na polu *STATION*, klawisz Q służy do przeszukania stacji, a klawisz W służy do przejęcia jej na własność.

4. Najważniejsze klasy i funkcje

W projekcie użyliśmy wzorca projektowego „Stany”, dzięki czemu jesteśmy w stanie podzielić kod. W naszym przypadku mamy osobny stan dla „menu”, „oczekiwanie na rozpoczęcie rozgrywki” oraz stan samej rozgrywki.

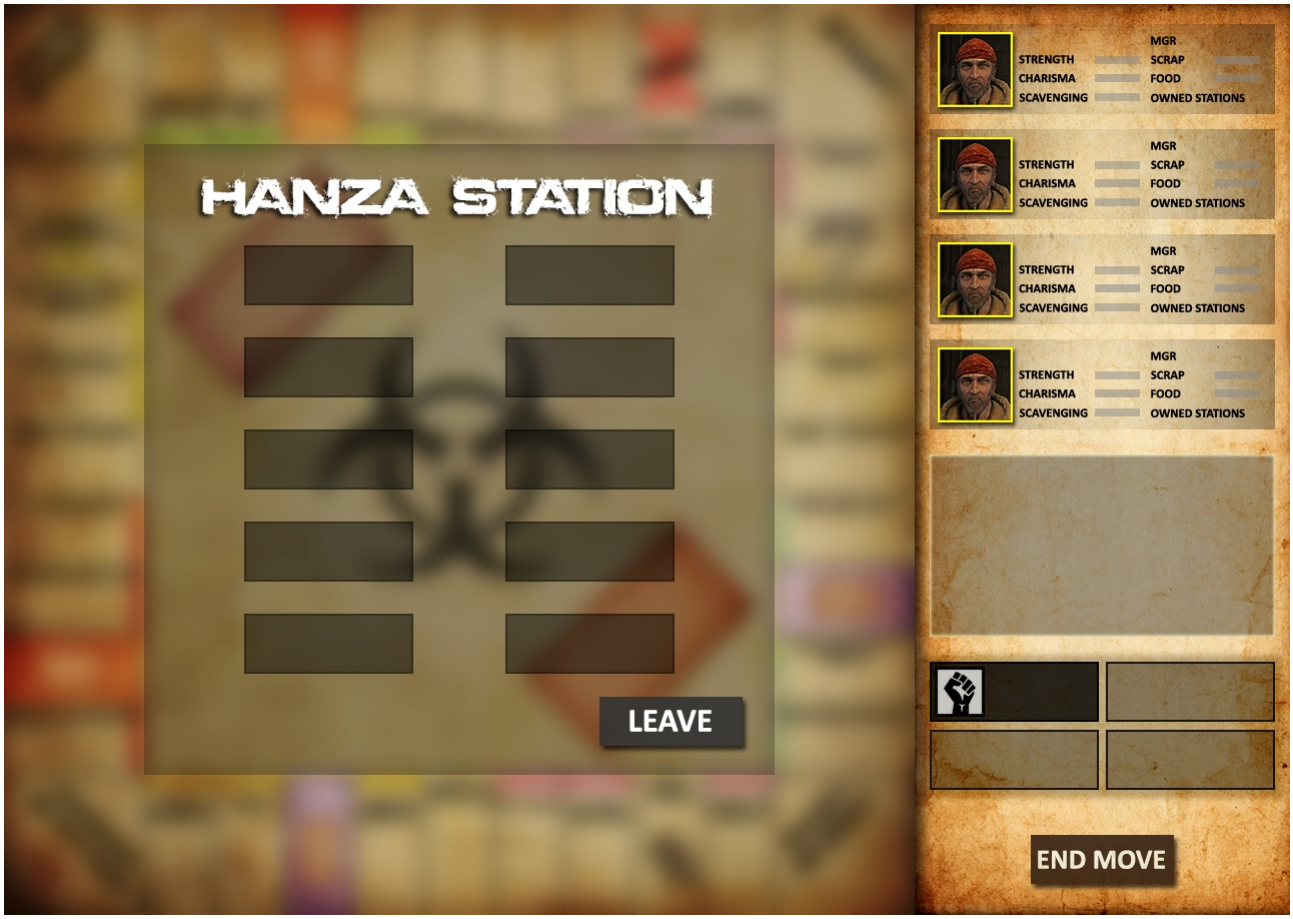
- **State** – klasa bazowa, która służy tylko do przesyłania przez klasy potomne takie jak Game State.
- **Game State** – najważniejsza klasa stanu rozgrywki. W niej zawarte są odwołania do innych klas, oraz zachodzi tam główna pętla gry.
- **Enemy** – zawiera dane o przeciwnikach oraz ładuje ich stan.
- **Player** – przechowuje dane o graczu.
- **Fight** – klasa symulująca klasę statyczną – służy do obliczania wyniku starcia gracza z przeciwnikiem.
- **GUI** – odpowiada za wyświetlanie graficznego interfejsu użytkownika na planszy.
- **Packet** – przechowuje dane o pojedynczym pakiecie, który zostaje wysłany do hosta, a potem rozesłany innym graczom. Posiada również metody pakujące i rozpakowujące pakiet.
- **TextureManager** – symuluje klasę statyczną – odpowiada za przechowywanie i ładowanie tekstur
- **SpriteManager** – symuluje klasę statyczną – odpowiada za przechowywanie i ładowanie sprite'ów.
- **Fields** – klasa bazowa służąca do przechowywania danych oraz logiki dla poszczególnych pól na planszy. Każdy typ pola jest własną klasą, np. ShopField, która służy do wyświetlania sklepu, jeśli gracz stanie na tym polu.
- **MainMenuState** – najważniejsza klasa dla stanu Menu – służy do wyświetlania głównego menu gry.
- **ServerMultiplayer** – klasa służy do stworzenia serwera i rozpoczęcia nasłuchiwania na połączenia od klientów.
- **ClientMultiplayer** – służy do łączenia się z serwerem.
- **ClientWaitingState** oraz **ServerWaitingState** – stan oczekiwania. Serwer nasłuchuje kolejnych połączeń, a połączony klient czeka na rozpoczęcie rozgrywki.
- **ServerGameState** – stan serwera, który działa podczas rozgrywki. Służy do odbierania pakietów od graczy i rozsyłania ich pozostałym.

5. Podział prac

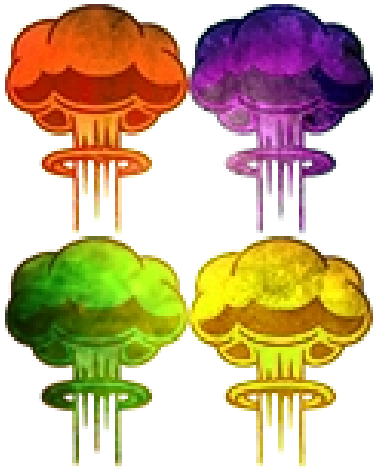
Ze względu na różnorodność zadań oraz ich ilość, podzieliliśmy obowiązki w taki sposób, że Emil Nowak zajął się grafiką oraz stanem przed rozgrywką, natomiast Kacper Obara zajął się główną rozgrywką oraz kwestiami sieciowymi.

Pomysł oraz klasy związane z rozgrywką zostały obmyślone wspólnie.

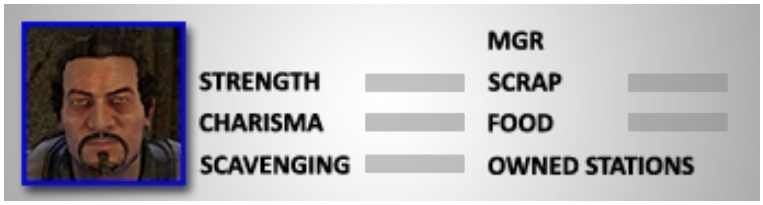
6. Plan na ukończenie gry



Pole sklepu, bez aktywnych przedmiotów do kupienia, wraz z interfejsem użytkownika.



Znaczniki przejęcia pola.



Statystyki gracza w formie ostatecznej mają mieć postać graficzną.