Projekt Programowanie w Języku Java

Politechnika Świętokrzyska

Temat:	
Autobattler	
Zespół:	Grupa:
Mróz Piotr	2ID12B

Wstęp

Założeniem projektu było stworzenie gry, w której gracze na zmianę wykonują swoje tury, a celem gry jest pokonanie swojego przeciwnika.

W każdej turze gracze mogą:

- Kupić postać
- Lub pominąć turę

Na początku rundy gra oblicza obrażenia otrzymane od postaci gracza przeciwnego, stan złota, wartość obrony oraz ilość pozostałych punktów życia obydwu graczy.

Drogę do zwycięstwa można wywalczyć wieloma taktykami takimi jak np. Budowanie silnej obrony co niweluje obrażenia przeciwnika lub atakowanie przeciwnika z poświęceniem własnej obrony. Można też szukać złotego środka pomiędzy atakiem i obroną, gdyż każda rozgrywka jest inna i to od nas zależy jaką taktykę zastosujemy.

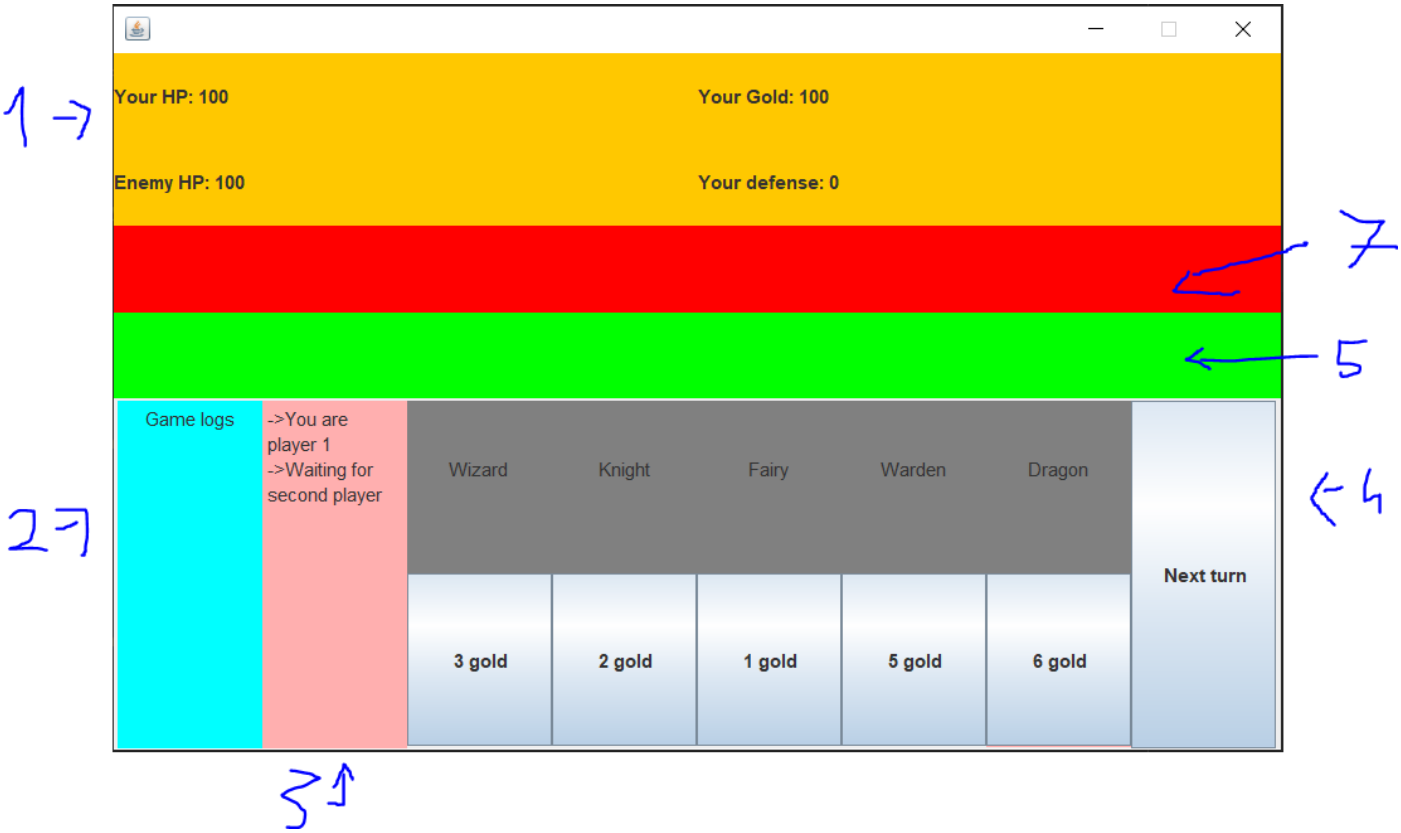
Mechanika

W grze można kupić następujące postacie

Postać	Wartość ataku	Wartość obrony
Wizard	3	1
Knight	2	2
Fairy	1	1
Warden	1	5
Dragon	4	5

Co turę gracz może dokonać zakupu postaci lub pominąć turę na czym zyska 2szt Złota. Po wykonaniu akcji automatycznie wykonuje się obliczanie obrażeń, złota, oraz pozostałych punktów zdrowia graczy.

Graficzny Interfejs Użytkownika



1 - Pasek statusu. Zawiera takie informacje takie jak wartość życia obydwu graczy, złoto oraz wartość obrony gracza.

2 - GameLogs. W tym miejscu wyświetlają się komunikaty związane z grą takie jak ilość otrzymanych obrażeń.

3 – Informacje systemowe. Wyświetla status połączenia z serwerem.

4 – Przyciski zakupu i pominięcia tury. W tym miejscu gracz może dokonać zakupu postaci lub pominąć turę.

5– Pole postaci gracza.

7– Pole postaci przeciwnika.

Aspekt Techniczny

Aby ułatwić pracę postanowiłem całość kodu podzielić na 4 pakiety(packages):

- Models (klasy gracz oraz postaci)
- Services (zawiera usługi wykorzystane w grze takie jak GameServer oraz Client)
- Shared (Współdzielone, klasa GameLogic)
- Oraz UI (Klasa zawierająca całość interfejsu graficznego)

UWAGA! Sprawozdanie zawiera powierzchowne opisanie kodu źródłowego, głębsze wyjaśnienia znajdują się w załączonej dokumentacji.

Klasy i ich opisy

GUI

Poniżej prezentuję drzewko GUI pod względem podziału na layout'y. Każdy panel posiada layout typu Grid.

- Frame
    - Container
        - Upper
            - StatusBar
            - Battlerground

        - MyHeroesOnField
        - enemyHeroesOnField
    - Lower
        - GameLogs
        - SystemInfo
        - HeroBuy1
            - Info1
            - Button1
        - HeroBuy2
            - Info2
            - Button2
        - HeroBuy3
            - Info3
            - Button3
        - HeroBuy4
            - Info4
            - Button4
        - HeroBuy5
            - Info5
            - Button5
        - HeroBuy6
            - Info6
            - Button6

Przykład kodu

```
public void setUpUi(){
    this.setSize(width, height); // basic configuration
    this.setTitle("Simple Autobattler");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setResizable(false);

    frame.add(container);
    container.setLayout(new GridLayout( rows: 2, cols: 0)); // adding grid layout to our container
    upper.setLayout(new GridLayout( rows: 2, cols: 1));
    lower.setLayout(new GridLayout( rows: 1, cols: 6));

    battleground.setLayout(new GridLayout( rows: 2, cols: 1));

    heroBuy1.setLayout(new GridLayout( rows: 2, cols: 1));
    heroBuy2.setLayout(new GridLayout( rows: 2, cols: 1));
    heroBuy3.setLayout(new GridLayout( rows: 2, cols: 1));
    heroBuy4.setLayout(new GridLayout( rows: 2, cols: 1));
    heroBuy5.setLayout(new GridLayout( rows: 2, cols: 1));
    heroBuy6.setLayout(new GridLayout( rows: 1, cols: 1));

    statusBar.setLayout(new GridLayout( rows: 2, cols: 1));

    container.add(upper);
    container.add(lower);


    gameLogs.setText("Game logs");
    gameLogs.setBackground( Color.CYAN);
    gameLogs.setEditable(false);
    //gameLogs.set
```

Powyższy kod przedstawia funkcję setUpUi która inicjalizuje część zmiennych i przypisuję im layout’y.

GameServer

Struktura:

1) Klasa GameServer
    a) Konstruktor GameServer
    b) Metoda acceptConnections
    c) Klasa wewnętrzna serverSideConnection
        i) Konstruktor klasy
        ii) Metoda closeConnections
        iii) Metoda run
        iv) Metoda sendButton

1.GameServer - (Klasa – server) odpowiedzialną za serwerową część działania gryKlasę tę należy uruchomić w pierwszej kolejności

A) Konstruktor domyślny klasy serwer. Nie przyjmuje żadnych parametrów, odpowiada za stworzenie socket'a

B) Metoda odpowiedzialna za przyjmowanie połączeń

C) Klasa wewnętrzna, odpowiedzialna za zarządzanie połączeniami ze strony serwera

I) Nie przyjmuje żadnych parametrów, odpowiada za incijalizację połączenia

II) Metoda odpowiada za zamykanie połączeń z serwerem

II) Metoda odpowiedzialna za zarządzanie przesyłem danych.

IV) Metoda wysyła wciśnięty przycisk(Jbutton)

Przykład kodu

```java
/**
 * Metoda odpowiedzialna za przyjmowanie połączeń
 */
public void acceptConnections(){
    try{
        while(connectedPlayers < 3) {
            Socket soc = serverSocket.accept();
            ++connectedPlayers;
            ServerSideCon ssc = new ServerSideCon(soc, connectedPlayers);
            System.out.println("connected players : " + connectedPlayers);

            if (connectedPlayers == 1) {
                player1 = ssc;
            } else {
                player2 = ssc;
            }
            Thread th = new Thread(ssc); // Initialize new thread handling Server
            th.start();
        }
    } catch(IOException e) {
        System.out.println("Exception form AcceptConnections : " + e);
    }
}
```

Obraz przedstawia klasę acceptConnections, która czeka na połączenie. Po czym inkrementuje licznik graczy w grze, tworzy nowy obiekt klasy ServerSideConnections który pozwala na obsługę połączeń oraz przypisuje obiekt do zmiennej player1 lub player2. Na koniec przypisujemy wykonanie zadania do wątku th. Całość może skutkować wyrzuceniem wyjątku e.

Game

Struktura:

- **2)** Klasa Game
  - **a)** Konstruktor Game
  - **b)** Metoda handlePlayer
  - **c)** Metoda HandleTurn
  - **d)** Metoda buttonHandler

2) Najważniejsza klasa tej gry, w niej wykonują się wszystkie znaczące operacje

A) Inicjalizuje podstawowe zmienne i przypisuje je do poszczególnych elemenów GUI

B) Metoda odpowiedzialna za zainicjalizowanie klasy gracza

C) Metoda odpowiedzialna za komuniakcję GUI – Player sprawdza czy gracz ma wystarczającą ilośc złota. Blokuje/odblokowuje przyciski. Sprawdza czy jest koniec gry Tworzy okienko Zwycięstwo/Porażka

D) Metoda "nasłuchuje" wciśniętych przycisków na podstawie czego wykonuje operacje takie jak obliczanie obrażeń,obrony tworzy nowy wątek któremu nakazuje wykonać funkcję handleTurn.

Przykład kodu

```
public void buttonHandler(){
    ActionListener al = e -> {
        JButton buttonPressed = (JButton) e.getSource();
        String buttonText = buttonPressed.getText();

        int whichObject = GameLogic.getPressedButton(buttonText); // id of pressed
        ui.gameLogs.setText("you bought " + champs[whichObject]);
        playerInstance.gold -= GameLogic.getHeroPrice(buttonText);
        ui.GOLD.setText("Your gold: " + playerInstance.gold);

        ui.gameLogs.setText("You clicked button #" + whichObject+  "Waiting for pla

        ui.isMyTurn = !ui.isMyTurn;
        GameLogic.toggleButtons(ui, playerInstance);

        playerInstance.listOfHeroes.add(whichObject);
        GameLogic.showHeroes(playerInstance, ui);
        GameLogic.checkIfEnoughGold(playerInstance, ui);

        playerInstance.defense = GameLogic.calculateDefense(playerInstance);
        playerInstance.enemyDefense = GameLogic.calculateEnemyDefense(playerInstanc
        playerInstance.health -= GameLogic.calculateDMG(playerInstance);
        playerInstance.enemyHealth -= GameLogic.calculateEnemyHealth(playerInstance

        ui.HEALTH.setText("Your hp: "+playerInstance.health);
        ui.gameLogs.setText("You took " + GameLogic.calculateDMG(playerInstance) +
        ui.defense.setText("Your defense: \n" + playerInstance.defense);
        ui.ENEMYHEALTH.setText("Enemy HP: \n" + playerInstance.enemyHealth);

        if(playerInstance.health < 1){
            JFrame defeat = new JFrame();
            defeat.setSize( width: 200, height: 70);
            defeat.setResizable(false);
```

Funkcja buttonHandler zawiera w sobie wyrażenie lambda ActionListener, która ”nasłuchuje” wciśniętych przycisków. Kiedy gracz wciśnie przycisk gra:

- ustala jaki przycisk to był
- Oblicza wydane złoto
- Blokuje przyciski
- Dodaje zakupioną postać do listy bohaterów gracza
- Wyświetla postać
- Zmienia logi w grze
- Oraz sprawdza czy jest to koniec gry

GameLogic

Struktura:

3) Klasa GameLogic
- a) Metoda getPressedButton
- b) Metoda getHeroPrice
- **c)** Metoda showHeroes
- **d)** Metoda checkIfEnoughGold
- **e)** Metoda toggleButtons
- **f)** Metoda calculateDMG
- **g)** Metoda calculateEnemyHealth
- **h)** Metoda calculateDefense
- **i)** Metoda calculateEnemyDefense

3) Klasa ta zawiera metody odpowiedzialne za logikę gry m.in. obliczają dokładnie ilość życia, obrażeń bohatera i przeciwnika

A) Metoda pobiera treść wciśniętego przycisku i zwraca jego ID. Zastosowano tu wyrażenie lambda

B) Metoda pobiera treść wciśniętego przycisku i zwraca jego koszt zakupu bohatera. Zastosowano tu wyrażenie lambda

C) Metoda wyświetla postacie gracza po jego stronie pola bitwy

D) Sprawdza czy gracz ma wystarczającą ilość złota na zakup postaci, metoda wywoływana jest na początku każdej rundy

E) Metoda Przełącza przyciski na początku rundy na aktywne

F) Metoda oblicza wartość obrażeń gracza iterując listę jego postaci

G) Metoda oblicza wartość hp przeciwnika

H) Metoda oblicza wartość obrony gracza iterując listę jego postaci

I) Metoda oblicza wartość obrony przeciwnika iterując listę jego postaci

Przykład kodu

```java
 * Metoda Przełącza przyciski na poczatku rundy na aktywne
 * @param playerInstance Player : instancja gracza
 * @param ui GUI : instancja klasy GUI
 */
public static void toggleButtons(GUI ui, Player playerInstance){
    if(ui.isMyTurn){
        ui.toggleButtonsEnabled();
        checkIfEnoughGold(playerInstance, ui);
    }else{
        ui.toggleButtonsNotEnabled();
    }
}

/**
 * Metoda oblicza wartość obrażeń gracza iterujac listę jego postaci
 * @param playerInstance Player : instancja gracza
 * @return obrazenia gracza w tej turze(INT)
 */
public static int calculateDMG(Player playerInstance){
    int dmgThisTurn = 0;
    for(Hero x : playerInstance.enemyHeroesOnField){
        dmgThisTurn += x.Attack;
    }
    return Math.max(dmgThisTurn - playerInstance.defense, 0);
}

/**
 * Metoda oblicza wartość hp przeciwnika
 * @param playerInstance Player : instancja gracza
 * @return hp przeciwnika
 */
public static int calculateEnemyHealth(Player playerInstance){
    AtomicInteger dmgThisTurn = new AtomicInteger();
```

Metoda toggleButtons – Najpierw sprawdza czy jest to tura gracza, jeśli tak gra odblokuje nam możliwość korzystania z przycisków, po czym oblicza czy stać nas na zakup każdej postaci za pomocą funkcji checkIfEnoughGold()

Metoda calculateDMG() - metoda ta iteruje listę postaci przeciwnika w celu policzenia całkowitych obrażeń w tej turze po czym oblicza deltę pomiędzy obrażeniami przeciwnika, a naszą obroną. Jeśli nasza obrona ma większą wartość niż wartość ataku przeciwnika to nie otrzymamy obrażeń.

Hero

Struktura:

4) Klasa Hero
    a) Konstruktor klasy

Przykład kodu

```java
package com.company.main.models;

/**
 * <h1>Hero</h1>
 * Klasa ta tworzy bohatera zawiera takie dane jak wartość jego ataku lub obrony
 * @author  Piotr Mróz
 * @since   2020-06-17
 * */
public class Hero {
    public String name;
    public int Attack;
    public int Defense;

    /**
     * Metoda oblicza wartość obrażeń gracza iterując listę jego postaci
     * @param name String : nazwa bohatera
     * @param a Int : atak bohatera
     * @param d Int : obrona bohatera
     */
    public Hero(String name, int a, int d){
        this.name = name;
        this.Attack = a;
        this.Defense = d;
    }
}
```

Client

Struktura:

5) Klasa Client
    a) Metoda connectoToServer
    b) Klasa wewnętrzna ClientSideCon
        i) Konstruktor klasy
        ii) Metoda sendButtonPressed
        iii) metoda reciveButtonPressed

5) Klasa - Klient odpowiedzialną za część klienta działania gry. Klasa ta jest wywoływana z klasy gracza(Player)
    A) Metoda tworzy nowe połączenie z serwerem
    B) Klasa wewnętrzna, odpowiedzialna za zarządzanie połączeniami ze strony klienta
        III) konstruktor, tworzy nowe połączenie z serwerem
        IV) Metoda wysyła ID wciśniętego przycisk(Jbutton)
        V) Metoda odbiera wciśnięty przycisk(Jbutton)

Przykład kodu

```
public ClientSideCon() {
    try {
        //get data streams
        socket = new Socket( host: "localhost",  port: 6968);
        dataIn = new DataInputStream(socket.getInputStream());
        dataOut = new DataOutputStream(socket.getOutputStream());

        player = dataIn.readInt();
        if(player == 1){
            enemy = 2;
        } else {
            enemy = 1;
        }
        //System.out.println("Connected to server #player : " + player + "\r
    } catch (IOException e) {
        System.out.println("Exception thrown from CSC" + e);
    }
}

/**
 * Metoda wysyła wciśnięty przycisk(Jbutton)
 * @param data Int : wciśnięty przycisk.
 * @exception IOException On input error.
 * @see IOException
 */
public void sendButtonPressed(int data){
    try{
        dataOut.writeInt(data);
        dataOut.flush();
    }catch(IOException e){
        System.out.println("Exception formsenbuttonclient : " + e);
    }
}
```

Konstruktor ClientSideCon najpierw tworzy nowe połączenie z portem 6968, po czym inicjalizuje dwa strumienie danych (dataIn, dataOut). Odbieramy od serwera pierwszą wiadomość która oznacza jaki ID gracza został nam przypisany.
Metoda sendButtonPressed wysyła ID wciśniętego przycisku.

Player

Struktura:

6) Klasa Player
    a) Konstruktor Player
    b) Metoda initHeroes

6) Klasa zawiera wszystkie informacje niezbędne do obsługi gracza
    A) konstruktor, inicjuje takie zmienne jak hp, gold, defense
    B) metoda inicjalizuje listę(Hero) postaci

Przykład kodu

```
public Player(int hp, int g){
    this.health = hp;
    this.gold = g;
    this.enemyHealth = 100;
    this.defense = 0;
    this.enemyDefense = 0;

    initHeroes();
    cl = new Client();
    cl.connectToServer();
    playerNumber = cl.player;
    //System.out.println("Im player no: " + playerNumber);
}

/**
 * metoda inicjalizuje listę(Hero) postaci
 */
public void initHeroes(){
    int i;
    for (i = 0 ; i< 5; ++i){
        switch (i) {
            case 0 -> {
                Hero wizard = new Hero( name: "Wizard",  a: 3,  d: 1);
                Heroes.add(wizard);
            }
            case 1 -> {
                Hero Knight = new Hero( name: "Knight",  a: 2,  d: 2);
```

Konstruktor Player() inicjalizuje zmienne health, gold, enemyHealth, defense, enemyDefense
Po czym wywołuje metodę initHeroes, oraz ustanawia nowe połączenie “cl”, korzystają z klasy Client oraz ClientSideCon.

Metoda initHeroes inicjalizuję listę (typu Hero) postaci w grze takich jak wizard, knight.

Main

Struktura:

7) Klasa Main

Przykład kodu

```java
package com.company.main;

import com.company.main.services.Game;


public class Main {
    /**
     * <h1>Main</h1>
     * Jest to funkcja główna z której wywołujemy klasę GAME odpowiedzialną za działani
     * <p>
     * @param args Unused.
     * @author  Piotr Mróz
     * @since   2020-06-17
     */
    public static void main(String[] args) { Game run = new Game(); }
}
```

Klasa ta za pomocą metody main jest punktem startowym całej gry.

Screenshot'y z gry

Wnioski

Projekt okazał się być znacznie trudniejszy do wykonania niż początkowo planowałem. Największe problemy sprawiała mi synchronizacja danych między graczami, co na wcześniejszych etapach produkcji skutkowało na losowym blokowaniu się wszystkich przycisków. Dużym problemem też było odpowiednie ulokowanie wszystkich elementów interfejsu graficznego przy wykorzystaniu biblioteki swing. Najbardziej żałuję że nie udało mi się zaimplementować wyświetlania graficznego postaci w Panelach należących do panelu battleground. Uważam że

dzięki temu gra mogła by znacznie zyskać na estetyczności. Poczatkowo też zastanawiałem się na dodatnie czarów do gry , ale nie zalazłem sposobu na sensowne wykorzystanie tego pomysłu.

Podsumowując projekt był dla mnie dużym wyzwaniem, ale nauczyłem się dzięki niemu takich rzeczy jak biblioteka swing, architektura Klient-Serwer oraz zarządzanie wielowątkowością.