Android aplikacija logičke igre stabla i šatori

Kukolj, Igor

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:976750

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-08**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij Informatike

ANDROID APLIKACIJA

LOGIČKE IGRE STABLA I ŠATORI

Završni rad

Igor Kukolj

Osijek, 2018 godina.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 21.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Igor Kukolj
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4480, 20.09.2017.
OIB studenta:	91851367679
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ivan Aleksi
Član Povjerenstva:	Doc.dr.sc. Anita Katić
Naslov završnog rada:	Android aplikacija logičke igre stabla i šatori
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada	Student će izraditi aplikaciju u Android sustavu za logičku zagonetku stabla i šatori. Sumentor: Alfonzo Baumgartner
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 29.09.2018.

Ime i prezime studenta:	Igor Kukolj
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4480, 20.09.2017.
Ephorus podudaranje [%]:	3 %

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija logičke igre stabla i šatori**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1.UVOD **1**

2. KORIŠTENE TEHNOLOGIJE **2**

2.1 Java **2**

2.2 LibGDX **2**

2.3 Android Studio **2**

2.4 JSON Format **3**

2.5 XML Format **3**

2.6 GIMP **3**

3. IGRANJE IGRE STABLA I ŠATORI **4**

3.1 Pravila igre **4**

3.2 Postupak rješavanja zagonetke „Stabla i Šatori“ **5**

4.RAZVOJ APLIKACIJE **8**

4.1. Skiciranje aplikacijskih zaslona **8**

4.2 Izrada konstrukcije grafičkih elemenata za igru **8**

4.3 Osnovna arhkitektura i programiranje zaslona **9**

4.4 Dizajn i programiranje klasa koje predstavljaju grafičke elemente **15**

4.5 Programiranje logičkih klasa za predstavljanje poslovne logike **15**

5.ZAKLJUČAK **20**

LITERATURA **21**

SAŽETAK **22**

ŽIVOTOPIS **24**

PRILOZI (DVD) **25**

1.UVOD

Cilj ovog završnog rada je izrada logičke igre stabla i šatori koja se može igrati na više platformi (desktop, mobile, web). Ova igra je napravljena koristeći cross-platform framework LibGdx. Za programiranje se koristio Android Studio IDE, glavna platforma za ovu igru su mobilni uređaji. U 2. poglavlju se ukratko opisuju tehnologije korištene za izradu ove aplikacije. Za programiranje je korišten programski jezik Java sa softverskim okvirom LibGDX. Softversko okruženje u kojem je igra programirana je Android Studio. Za generiranje levela korišteni su podatci spremljeni u JSON formatu. Za spremanje podataka o konfiguraciji aplikacije korišten je XML format. Glavna konfiguracija je AndroidManifest .xml u kojem je spremljen: naslov igre, ikona, tema, orijentacija zaslona itd. Kreiranje konstrukcije grafičkih elemenata(slika) je napravljeno koristeći program GIMP. U GIMP-u su kreirane slike za predstavljanje konstrukcijskih elemenata u igri(stabla, šatori, zasloni, gumbi, teksture). U 3. poglavlju se objašnjavaju pravila igre i način na koji se ova igra treba igrati. Uz pravila se vide slike koje nam daju vizualni dojam o pravilima koja se trebaju poštivati. U 4. poglavlju se vidi detaljan postupak razvoja igre stabla i šatori. Taj postupak se sastoji od nekoliko važnih koraka a to su skiciranje zaslona (radi odlučivanja rasporeda elemenata unutar svakog zaslona), izrada konstrukcijskih elemenata, dizajniranje programske arhitekture, pisanje logičkog koda za realizaciju aplikacije i spremanje podataka koji opisuju levele za igranje.

2. KORIŠTENE TEHNOLOGIJE

2.1 Java

Java je objektno orijentirani jezik razvijen u tvrtki Sun Microsystem. Neki od značajnijih stručnjaka koji su radili na razvijanju ovoga jezika su Patrick Naughton i James Gosling.Razvijanje ovog jezika je krenulo 1991. g. a prva verzija jave je objavljena 1995. Jedna od velikih prednosti ovog jezika je to da se kod napisan u javi može izvoditi na svim sustavima koje imaju JVM (Java Virtual Machine) .[1] Tu Java ima veliku prednost u odnosu na jezike poput C i C++ koje su ovisne o platformi i moraju se prilagoditi operacijskom sustavu na kojem se izvode. Java je među jezicima koji se najviše koriste, te se procjenjuje da javu koristi 7-10 milijuna korisnika. Važno je napomenuti da se java može upotrebljavati za mnoge svrhe: web aplikacije, mobilne aplikacije, desktop aplikacije, serveri itd...

2.2 LibGDX

LibGDX je besplatan softverski okvir(framework) koji se koristi za pravljenje igri za više-platformsko okruženje. Softverski okvir je napisan u Java programskom jeziku ali neki dijelovi su napisani u C i C++ tako da dobiju bolje performanse softverskog okvira. [4] Korištenjem ovog softverskog okvira možemo praviti igre koje se mogu izvesti za više platformi: Windows, Linux, Android, iOS, Mac OS X i web preglednike koji imaju WebGL podršku. [5]

2.3 Android Studio

Android Studio je IDE (intergrated development enviroment) za android operativni sustav koji je napravljen na bazi od JetBrains i IntelliJ softverskog okruženja. Android Studio ima verzije za Windows, Linux i MacOs operacijske sustave. On je zamjena za dosadašnji primarni način razvoja mobilnih aplikacija koji je se izvodio korištenjem Eclipse-a s android dodatnim alatom. Android studio je najavljen 2013. g. od Google-a, prva verzija 1.0 je bila objavljena 2014. g. dok je trenutna verzija 3.1.3 iz lipnja 2018.[3] Tijekom razvijanja novijih verzija android studija zahtjevi za RAM memoriju se povećavaju sa svakom novom verzijom tako da trenutno primjerice treba 8 GB da studio normalno funkcionira, a zadnjih nekoliko godina bilo je dovoljno 2 ili 4GB RAM-a

2.4 JSON Format

JSON(JavaScript Object Notation) je format koji se najviše koristi za prijenos objektnih podataka koji je čitljiv korisnicima. Objekti unutar formata se sastoje od matrica i parova atribut-vrijednost ili bilo kojih drugih vrijednosti koje se mogu serijalizirati (U kodu implementirati serializable interface nad klasom). Ovo je često korišten format za podatke koji se asinkrono prijenose preko poslužitelja, služio je kao zamjena XML formatu koji je se prije JSON-a najviše koristio za ovakvu pohranu i prijenos. JSON datoteke imaju ekstenziju .json a njihov media tip je application/JSON..

2.5 XML Format

XML(Extensible Markup Language) je format koji ima definirana pravila za enkodiranje dokumenata tako da je čitljiv korisnicima. Glavni cilj XML formata je da bude jednostavan, općenit i koristan za prijenos podataka preko interneta. XML format ima UNICODE podršku za različite jezike. Upotrebljava se za prezentaciju podataka(spremanje podataka, gui sučelje kod android studia) i za prijenos podataka (API).

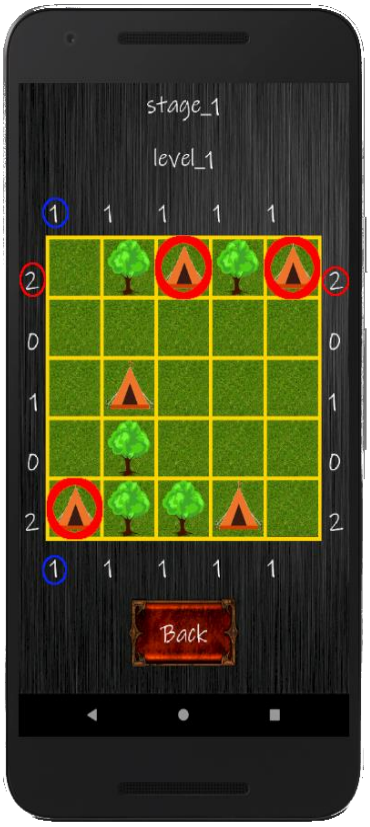
2.6 GIMP

Program korišten za manipuliranje, stvaranje i obradu rasterske grafike. Gimp je besplatan za upotrebu i ima mnoge mogućnosti koje ni plaćeni programi nemaju. Razvoj GIMP-A je započeo 1995. g. a prva verzija je objavljena 1996. g. On je u početku bio studentski projekt sa Kalifornijskog sveučilišta Berkley. Glavni programeri koji su u početku radili na ovom programu su Spencer Kimball i Petter Matis. GIMP je napisan u programskom jeziku C.

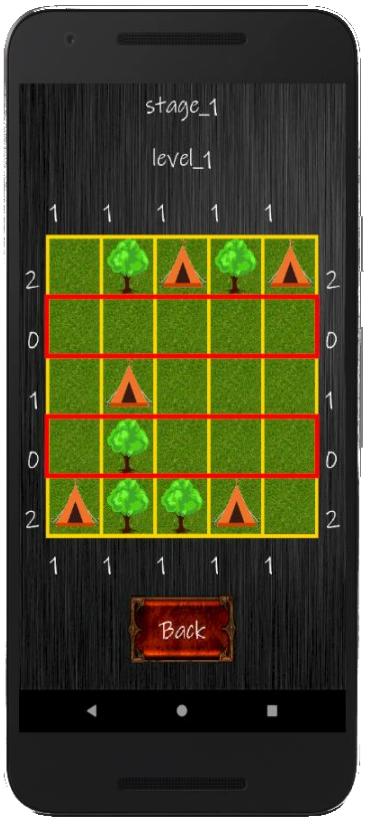
3. IGRANJE IGRE STABLA I ŠATORI

3.1 Pravila igre

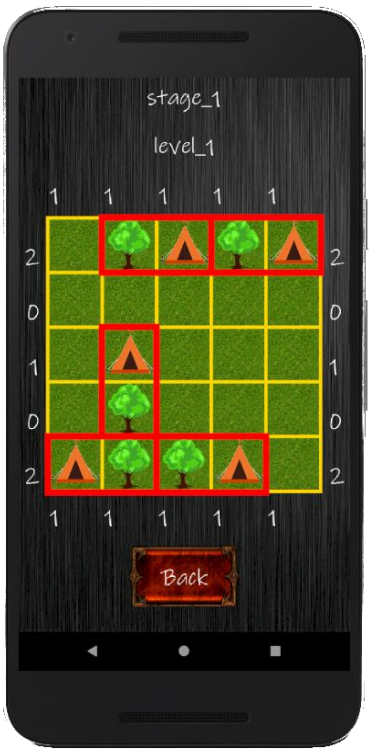
- Pronaći sve skrivene šatore.
- Svaki šator pripada samo jednom drvetu.
- Broj šatora i drveća je jednak.
- Svaki šator se može nalaziti vertikalno ili horizontalno od drveta tj. ne može biti u dijagonali prema slici 3.3.
- Šatori se ne smiju dodirivati horizontalno, vertikalno ili ukoso prema slici 3.4.
- Šator može biti između više drveća ali pripada samo jednom drvetu.
- Pomoćni brojevi horizontalno i vertikalno nam govore koliko šatora ima u određenom retku ili stupcu prema slici 3.1 i 3.2.



Sl. 3.1 Brojevi sa strane označavaju koliko ima šatora.



Sl. 3.2 Redak ili stupac sa brojem. ne sadrži šatore.



Sl. 3.3 Šatori su horizontalno ili vertikalno od stabla.

Sl. 3.4 Šatori se dodiruju.

3.2 Postupak rješavanja zagonetke „Stabla i Šatori“

Zagonetke ovog tipa se mogu rješiti koristeći postupak koji se sastoji od nekoliko koraka :

- 1.korak – Ispuniti sve redke/stupce koji sadržavaju 0 sa površinskim poljima.
- 2.korak – Polja koja horizontalno niti vertikalno ne graniče sa stablima ispuniti površinskim poljima.
- 3.korak – Ispuniti polja koja su ostala kao jedina mogućnost sa šatorima, to mogu biti 2 slučaja :
    - 1.slučaj Broj preostalih polja u retku/stupcu odgovara pomoćnom broju onda ih se može popuniti sa šatorima.
    - 2.slučaj Dva šatora se ne smiju dodirivati pa ako imamo dva susjedna mjesta prazna znamo da tu može biti samo 1 šator.

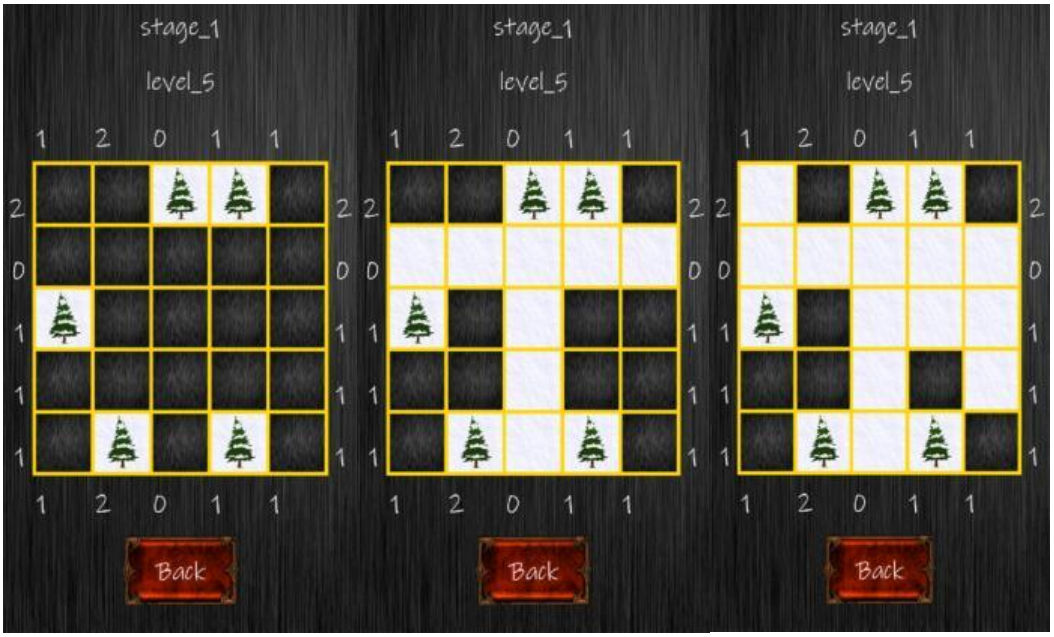
Nakon ovakvog zaključka gledamo da li broj preostalih praznih mjesta u tom retku/stupcu odgovara pomoćnom broju.

- 3.slučaj – Svako stablo mora imati šator, ako šator ima samo jedno prazno susjedno polje onda se ispuni sa šatorom.

- 4.korak – popuniti sva prazna susjedna polja oko šatora s površinskim poljima.
- 5.korak – popuniti stupac/redak s površinskim poljima ako broj postavljenih šatora u stupcu/retku jednak.pomoćnom broju.
- 6.korak – ponavljati korake 3-5 do kraja.

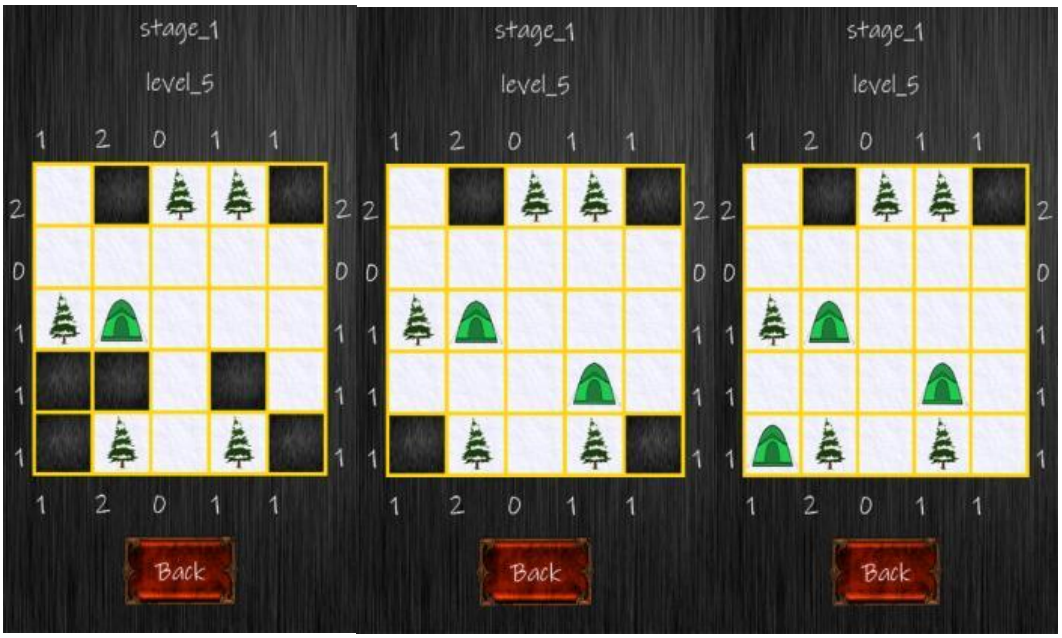
Potpun primjer rješavanja zagonetke

Za rješavanje ove zagonetke korišteni su prethodno definirani koraci.



Sl. 3.5 Početno stanje *Sl. 3.6 1.korak* *Sl. 3.7 2.korak*

Za ovaj primjer kreće se za zagonetkom iz prve grupe zadataka i imena level_5. Prema slici 3.5 vidi se početno stanje zagonetke koja je veličine 5x5 i sadrži 5 stabla. Prvo se upotrebljava 1.korak prema slici 3.6 zbog kojeg punimo drugi redak i treći stupac s površinskim poljima. Nakon toga upotrebljava se 2.korak zbog kojeg se ispunjavaju sva polja koja ne graniče sa stablima prema slici 3.7.



Sl. 3.8 3.korak, 1.slučaj *Sl. 3.9 6.korak prvi put* *Sl. 3.10 6.korak drugi put*

Poslije tog se upotrebljava 3.korak,1 slučaj prema slici 3.8 gdje se polje popuni sa šatorom zato što je jedino preostalo moguće polje za određeni redak. Nakon toga postupak se svodi na ponavljanje petlje od 3-5 koraka sve dok se ne dođe do dijaloga koji nam ukazuje da je level prijeđen .

Sl. 3.11 Prijeđen level

4.RAZVOJ APLIKACIJE

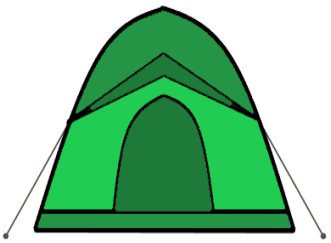
4.1. Skiciranje aplikacijskih zaslona

Prvi korak u razvoju ove aplikacije je bio odlučiti koliko zaslona je potrebno za ovu igru i kako će oni izgledati. Zaključeno je da će ova aplikacija imati 5 zaslona. Nakon toga napravljene su skice za svaki zaslon kako bi se došlo do odluke kako će izgledati raspored unutar svakog zaslona i koja će biti njegova funkcionalnost. Skice su rađene ručno na papiru i po njima je se došlo do daljnjih odluka vezanih za samu arhitekturu zaslona. Prilikom skiciranja zaslona došlo je se i do drugih odluka uključujući način na koji će se razdijeliti grafički elementi s obzirom na korisnikov odabir teme.

4.2 Izrada konstrukcije grafičkih elemenata za igru

Prilikom izrade konstrukcije grafičkih elemenata korišteni je program za crtanje GIMP i grafički tablet Huion 680S Proffesional. Neki od izrađenih grafičkih elemenata poput: šatora za temu zima, šatora za temu ljeto, ljetno drvo, zimsko drvo, jesensko drvo, blok snijeg itd. su bili korišteni kod implementacije za rješavanje logičkih zagonetki. Drugi grafički elementi uključujući : okvire unutar kojih se smješta zagonetka, tekstura pozadine, tekstura gumba, okvir gumba itd. su bili korišteni tijekom cijelog tijeka aplikacije tj. na mnogim mjestima a ne samo prilikom rješavanja zagonetke.

Sl. 4.1. Blok trava



Sl. 4.2. Šator

Sl. 4.3. Ljetno drvo

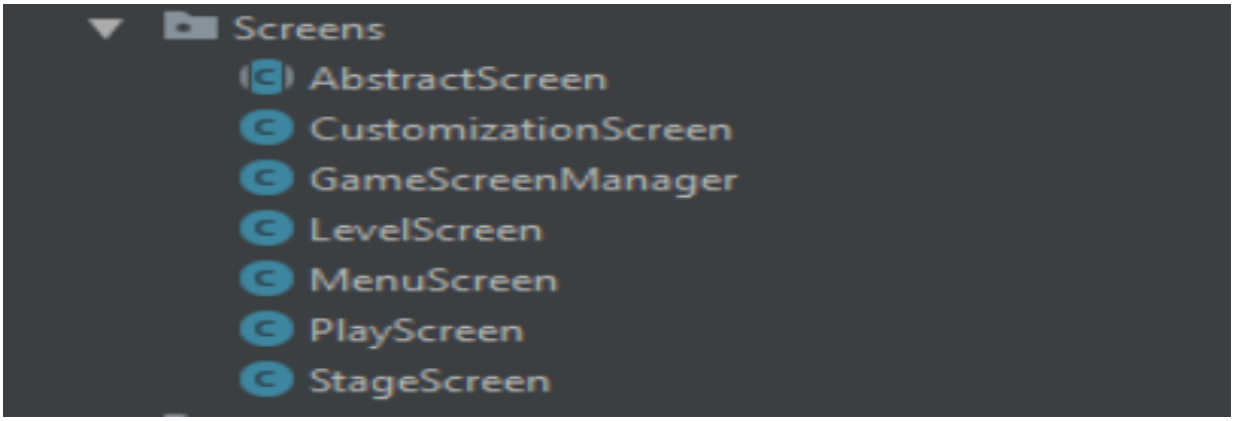
Sl. 4.4. Jesen drvo

Sl. 4.5. Zima drvo

Sl. 4.6. Šator2

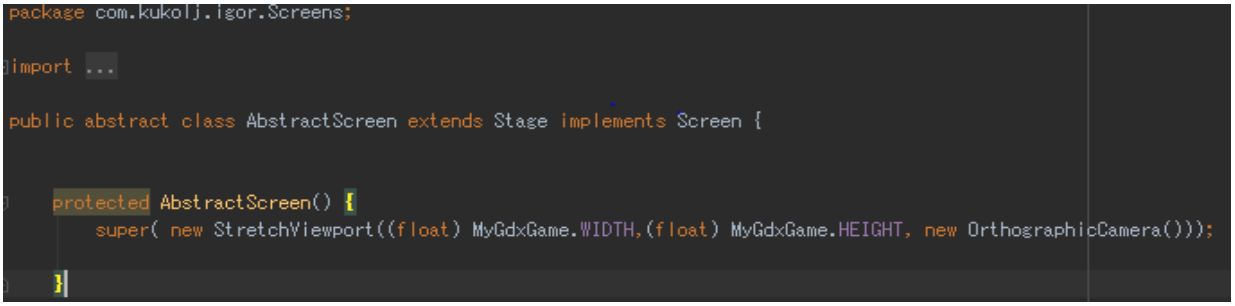
4.3 Osnovna arhkitektura i programiranje zaslona

Prilikom razvijanja ovakve aplikacije potrebno je prvo proučiti načine pisanje koda i sintaksu LibGDX Framework-a. Potrebno je razumjeti kontrolu zaslona i drugih elemenata unutar aplikacije. Dobar način za razumjeti logiku zaslona je eksperimentirati s arhitekturom dok se ne napravi dovoljno abstraktna arhitektura koja neće zakazati prilikom nadogradnje aplikacije.



Sl. 4.7. Paket koji sadrži sve zaslone

Prema slici 4.7 možemo vidjeti paket Screens u kojem se nalaze klase koje predstavljaju logiku za određeni zaslon i dvije klase AbstractScreen i GameScreenManager koje povećavaju razinu abstrakcije u kodu.



```
package com.kukolj.igor.Screens;

import ...

public abstract class AbstractScreen extends Stage implements Screen {

    protected AbstractScreen() {
        super( new StretchViewport((float) MyGdxGame.WIDTH,(float) MyGdxGame.HEIGHT, new OrthographicCamera()));

    }
```

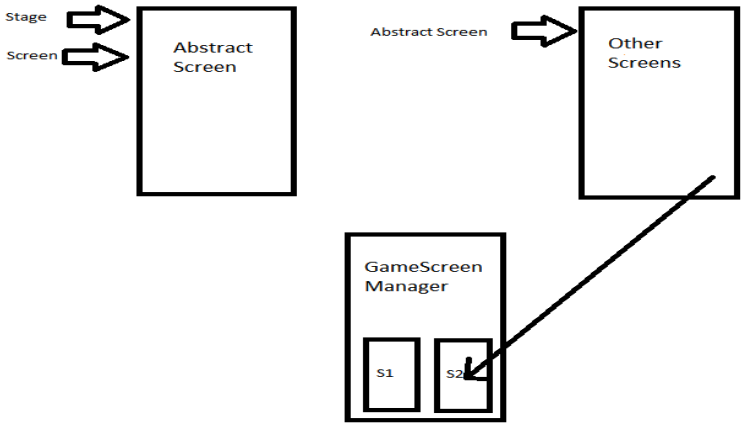
Sl. 4.8. Implementacija abstraktnog zaslona

Za ovaj projekt je prema slici 4.8 odlučeno koristiti Abstraktni zaslon koji će nasljeđivati klasu Stage i implementirati sučelje Screen. Stage daje mogućnost dodavanja grafičkih elemenata tj(Actora) na pozadinu dok Screen sučelje određuje metode koje svaki zaslon mora imati. AbstractScreen je apstrakna klasa koju svaki zaslon mora implementirati.

```
public AbstractScreen screenFactory(String screenKey) {
    AbstractScreen screen = null;
    if(screenKey.equals("MENU_SCREEN")){
        screen = new MenuScreen( gameScreenManager: this);
    }
    if(screenKey.equals("STAGE_SCREEN")){
        screen = new StageScreen( gameScreenManager: this);
    }
    if(screenKey.equals("LEVEL_SCREEN")){
        screen = new LevelScreen( gameScreenManager: this);
    }
    if(screenKey.equals("PLAY_SCREEN")){
        screen = new PlayScreen( gameScreenManager: this);
    }
    if(screenKey.equals("CUSTOMIZE_SCREEN")){
        screen = new CustomizationScreen( gameScreenManager: this);
    }
    return screen;
}
```

Sl. 4.9. Tvornica zaslona unutar GameScreenManager klase

GameScreenManager se brine kada će se koji zaslon aktivirati, sakriti ili obrisati. GameScreenManager klasa prema slici 4.9 ima tvorničku metodu koja kreira zaslone ovisno nizu slova koji joj se proslijedi.



Sl. 4.10. Arhitektura zaslona

```
playBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        gameScreenManager.set("STAGE_SCREEN");
    });
customizeBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        gameScreenManager.set("CUSTOMIZE_SCREEN");
    });
```

Sl. 4.11. Slušači za gumbe unutar MenuScreen klase

MenuScreen klasa sadrži logiku za prikazivanje zaslona, tekstura i gumba. Najvažnija zadaća ove klase je poslati korisnika na drugi zaslon prilikom pritiska na određeni gumb. Takva funkcionalnost je implementirana korištenjem slušača prema slici 4.11.

```
firstBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        Stats.getInstance().setCurrentStage("stage_1");
        gameScreenManager.set("LEVEL_SCREEN");
    });
secondBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        Stats.getInstance().setCurrentStage("stage_2");
        gameScreenManager.set("LEVEL_SCREEN");
    });
thirdBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        Stats.getInstance().setCurrentStage("stage_3");
        gameScreenManager.set("LEVEL_SCREEN");
    });
fourthBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        Stats.getInstance().setCurrentStage("stage_4");
        gameScreenManager.set("LEVEL_SCREEN");
    });
fifthBtn.addListener( (ClickListener) clicked(event, x, y) -> {
        Stats.getInstance().setCurrentStage("stage_5");
        gameScreenManager.set("LEVEL_SCREEN");
    });
```

Sl. 4.12. Slušači za gumbe unutar StageScreen klase

StageScreen klasa sadrži logiku za prikazivanje grafičkih elemenata. Zadaća ove klase je poslati korisnika na LevelScreen zaslon uz postavljanje odabrane skupine zadatku u statičku klasu Stats. Ova funkcionalnost je implementirana korištenjem slušača prema slici 4.12.

```
List<Level> levelList = Stats.getInstance().getCurrentStage().getLevelList();
double stepStart = 2.5;
for( final Level level: levelList){
    TextButton button = new TextButton(level.getName(), style);
    button.setBounds(xPos, (float) (heightSlice*stepStart),width,height);
    addActor(button);
    stepStart+=1.5;

    button.addListener( (ClickListener) clicked(event, x, y) -> {
            Stats.getInstance().setCurrentLevel(level.getName());
            gameScreenManager.set("PLAY_SCREEN");
    });
}
```

Sl. 4.13. Slušači za gumbe unutar LevelScreen klase

LevelScreen klasa kao i ostali zasloni sadrži logiku za prikazivanje grafičkih elemenata. Zadaća ove klase je poslati korisnika na PlayScreen zaslon uz postavljanje odabranog levela u statičku klasu Stats. Ova funkcionalnost je implementirana korištenjem slušača prema slici 4.12. Ova klasa dinamički stvara gumbe ovisno o broju dostupnih levela.

```
for(int i=0; i<tileArray.size(); i++){
    final com.kukolj.igor.Gameplay.Tile currentDataTile = currentTileList.get(i);
    final String currentDataStatus = currentDataTile.getStatus();
    final Tile currentGraphicalTile = tileArray.get(i);

    currentGraphicalTile.changeTextureList(currentDataStatus);
    addActor(currentGraphicalTile);

    currentGraphicalTile.addListener( (ClickListener) clicked(event, x, y) -> {
                currentDataTile.nextStatus();
                String newStatus = currentDataTile.getStatus();
                currentGraphicalTile.changeTextureList(newStatus);
                if(level.checkIfSolved()){
                    level.resetCurrentTileList();
                    Skin uiSkin = new Skin(Gdx.files.internal( path: "uiskin.json"));
                    Dialog dialog = new Dialog( title: "", uiSkin,  windowStyleName: "dialog") {
                        public void result(Object obj) {
                            if(obj.equals(true)) gameScreenManager.set("STAGE_SCREEN");
                        }
                    };
                    //dialog.scaleBy((float) 1.1);
                    dialog.scaleBy((float) 0.8);
                    float dialogW = (float) (dialog.getWidth()*1.8);
                    float dialogH = (float) (dialog.getHeight()*1.8);
                    float dialogX = getWidth()/2-dialogW/2;
                    float dialogY = getHeight()/2-dialogH/2;
                    dialog.setPosition(dialogX, dialogY);
                    dialog.text("Congratulations");
                    dialog.button( text: "OK",  object: true); //sends "true" as the result
                    addActor(dialog);
                }
    });
}
```

Sl. 4.14. Provjera završetka levela unutar PlayScreen klase

PlayScreen klasa provjerava da li je korisnik završio level. Ova funkcionalnost je implementirana prema slici 4.14.

```
public List<Tile> initializeTileArray(Board board, int boardSize, int gap, List<Integer>
horizonatlHelp, List<Integer> verticallHelp, List<Texture> textureList){

    List<Tile> tileArray = new ArrayList<Tile>();

    float boardX = board.getX();
    float boardY = board.getY();
    float boardDimension = board.getWidth();
    float dimension = (boardDimension-(boardSize+1)*gap)/boardSize;
    float startX = boardX;
    float startY = boardY + board.getHeight() - dimension;

    Label.LabelStyle textStyle = new Label.LabelStyle();
    textStyle.font = font;

    for(int i= 0;i<boardSize; i++){
        for(int j= 0;j<boardSize; j++){
            float x = startX+(j+1)*gap+j*dimension;
            float y = startY-(i+1)*gap-i*dimension;
            Tile tile = new Tile(x,y,dimension,dimension,textureList);
            tileArray.add(tile);

            //ADD VERTICAL HELPER TEXT
            if(j==0){
                String helpValue = verticallHelp.get(i).toString();
                Label text = new Label(helpValue, textStyle);
                //text.setFontScale((float) 0.5);
                text.setPosition(x-text.getWidth()-3*gap,y);
                addActor(text);
            }else if(j==boardSize-1){
                String helpValue = verticallHelp.get(i).toString();
                Label text2 = new Label(helpValue, textStyle);
                text2.setPosition(x+dimension+3*gap,y);
                addActor(text2);
            }

            //ADD VERTICAL HELPER TEXT
            if(i==0){
                String helpValue = horizonatlHelp.get(j).toString();
                Label text = new Label(helpValue, textStyle);
                //text.setFontScale((float) 0.5);
                text.setPosition(x,y+dimension+3*gap);
                addActor(text);
            }else if(i==boardSize-1){
                String helpValue = horizonatlHelp.get(j).toString();
                Label text2 = new Label(helpValue, textStyle);
                text2.setPosition(x,y-text2.getHeight()-3*gap);
                addActor(text2);
            }
        }

    }
    return tileArray;
```

Sl. 4.15. Generiranje grafike za rješavanje zagonetke

PlayScreen klasa je najvažnija klasa unutar ove aplikacije. Klasa dinamički generira level ovisno o podatcima o sadržaju i veličini levela. Funkcija za dinamički generiranje levela se može pogledati prema slici 4.15

Sl. 4.16. Menu zaslon

Sl. 4.17.Kustomizacijski zalson

Sl. 4.18. Stage zaslon

Sl. 4.19.Level zalson

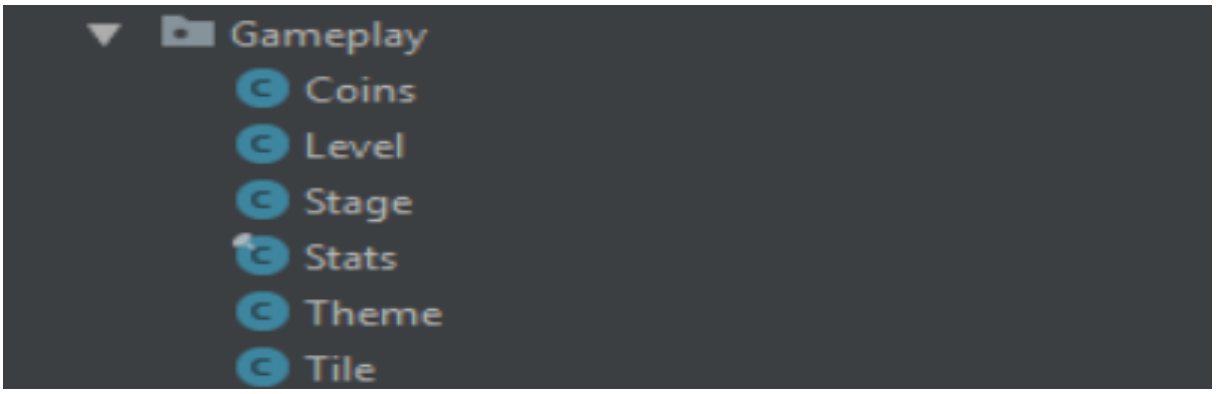
4.4 Dizajn i programiranje klasa koje predstavljaju grafičke elemente

Prema slici 4.20 napravljen je paket sprites u kojem se nalaze klase koje predstavljaju grafičke elemente koji će se koristiti prilikom generiranja zagonetke za rješavanje u PlayScreen zaslonu.. Svaka klasa se sastoji od X/Y koordinata, teksture, konstruktora, načina za crtanje na pozadinu i drugih metoda koje su specifične za svaku klasu.

Sl. 4.20. Sprites paket

4.5 Programiranje logičkih klasa za predstavljanje poslovne logike

Za logiku koja će se koristiti unutar igre napravljen je paket Gameplay koji sadrži poslovnu logiku ove aplikacije. Neke od klasa unutar ovog paketa predstavljaju logiku za grafičke elemente is sprites paketa (Tile,Theme). U ovom paketu se nalazi statička klasa Stats koja predstavlja trenutno stanje levela koji se rješavaju, služi kao privremena sesija koja traje dok je aplikacija upaljena u memoriji. Tu se nalaze i klase Level, Stage koje predstavljaju levele i skupine u koje su leveli podijeljeni.



Sl. 4.21. Gameplay paket

```
public final class Stats {

    private static final Stats INSTANCE = new Stats();

    private List<Stage> stageList;
    private Theme theme;

    private Stage currentStage;
    private Level currentLevel;

    private Stats() {
        stageList = JsonDataParser.getInstance().getStages();
        theme = new Theme( status: "SUMMER");
    }

    public static Stats getInstance() { return INSTANCE; }
```

Sl. 4.22. Varijable i konstruktor Stats klase

Stats klasa je veoma važna klasa iz Gameplay paketa koja služi kao sjednica koja traje dok se aplikacija ne restarta. Stats klasa u sebi sadrži sve skupine levela(Stage objekti), sve levele ,trenutnu temu, trenutni level i trenutnu skupinu u kojoj se taj level nalazi. Stats klasa je statična što znači da možemo koristiti njene atribute bez da je instanciramo, uz to ona sama sadrži svoju instancu što se vidi prema slici 4.22.

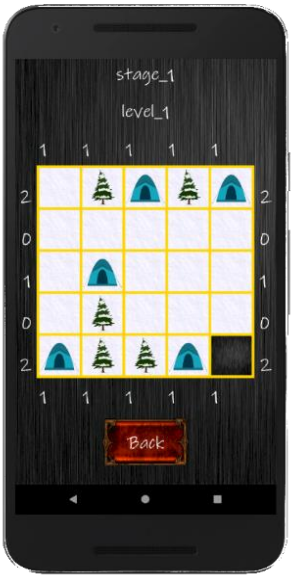
```
public void setTextureList(String themeKey) {
    List<Texture> tempTextureList = null;
    if(themeKey.equals("SUMMER")){
        textureList = Arrays.asList(new Texture( internalPath: "grass.png"),new Texture( internalPath: "summer_tree.png")
                ,new Texture( internalPath: "tent1.png"),new Texture( internalPath: "bg.png"));
    }
    if(themeKey.equals("AUTUMN")){
        textureList = Arrays.asList(new Texture( internalPath: "grass.png"),new Texture( internalPath: "autumn_tree.png")
                ,new Texture( internalPath: "tent1.png"),new Texture( internalPath: "bg.png"));
    }
    if(themeKey.equals("WINTER")){
        textureList = Arrays.asList(new Texture( internalPath: "snow.png"),new Texture( internalPath: "winter_tree.png")
                ,new Texture( internalPath: "tent2.png"),new Texture( internalPath: "bg.png"));
    }
}
```

Sl. 4.23. setTextureList metoda Theme klase

Klasa Theme koja se nalazi u Gameplay paketu koji služi kao spremnik podataka i tekstura koje se koriste za prikaz podataka prilikom igranja. U ovoj igri se trenutno nalaze 3 teme :

- Zima
- Ljeto
- Jesen

Theme klasa postavlja teksture ovisno o nizu znakova koji joj se proslijedi preko funkcije setTextureList prema slici 4.23. Svaka tema ima teksture koje bi trebale predstavljati godišnje doba. Aktivirana tema se sprema u klasu Stats prema kojoj se odlučuje koje će se teksture koristiti prilikom generiranja levela.



Sl. 4.24 Zimska tema *Sl. 4.25 Jesenska tema* *Sl. 4.26 Ljetna tema*

```
public class Stage {
    private String name;
    private List<Level> levelList;
    private Boolean completed;
    private Boolean owned;
    private int cost;
```

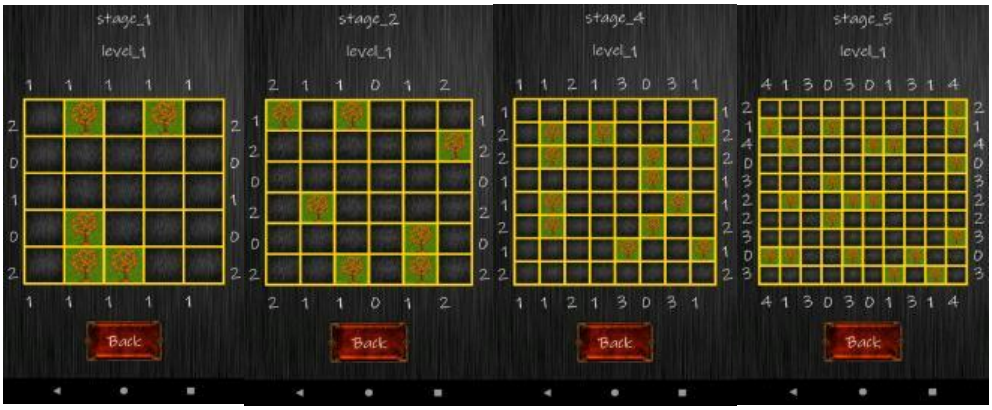
Sl. 4.27. Varijable Stage klase

Svrha Stage klase je da služi za instanciranje skupina levela koji su grupirani po različitim težinskim razinama. Stage klasa u sebi sadrži svoje ime, listu levela i completed vrijednost koja predstavlja da li je skupina levela obavljena. Varijable se mogu vidjeti prema slici 4.27. Postoji 5 težinskih razina u aplikaciji: Stage_1(5x5) leveli, Stage_2(6x6) leveli, Stage_3(7x7) leveli, Stage_4(8x8) leveli, Stage_5(10x10 leveli).

```
public class Level {
    private Boolean completed;
    String name;
    private int size;
    private List<Integer> verticalHelp;
    private List<Integer> horizontalHelp;
    private List<Tile> solution;
    private List<Tile> tileList;
    private List<Tile> currentTileListStatus;
```

Sl. 4.28. Varijable Level klase

Svrha Level klase je da služi za instanciranje levela u PlayScreen zaslonu za rješavanje logičke zagonetke. Level klasa u sebi sadrži svoje ime, listu levela, veličinu, pomoćne vrijednosti koje označavaju količinu šatora u stupcima i redcima, mapu koja predstavlja rješenje i mapu curretTileListStatus koja predstavlja trenutno stanje zagonetke. Varijable se mogu vidjeti prema slici 4.28. Postoji 5 vrsti levela u aplikaciji: 5x5, 6x6, 7x7, 8x8, 10x10.



Sl. 4.29. Prikaz levela različite težine

```
[
  [
    {"name" : "stage_1"},
    {
      "name" : "level_1",
      "size" : 5,
      "rowH" : [2,0,1,0,2],
      "columnH" : [1,1,1,1,1],

      "task" :    [0,2,0,2,0,
                   0,0,0,0,0,
                   0,0,0,0,0,
                   0,2,0,0,0,
                   0,2,2,0,0],

      "solution":    [1,2,3,2,3,
                      1,1,1,1,1,
                      1,3,1,1,1,
                      1,2,1,1,1,
                      3,2,2,3,1]
    },

    {
      "name" : "level_2",
      "size" : 5,
      "rowH" : [1,0,2,0,2],
      "columnH" : [1,2,0,1,1],

      "task" :    [0,2,0,0,0,
                   0,0,0,0,2,
                   0,0,0,0,0,
                   0,2,0,0,0,
                   2,0,0,0,2],

      "solution":    [3,2,1,1,1,
                      1,1,1,1,2,
                      1,3,1,1,3
```


Sl. 4.30. Prikaz spremljenih levela u .json formatu

Svaki level je grupiran u skupinu levela(stage). Skupina levela je grupirana u Json listu i svaki level je grupiran u posebnu listu unutar liste sa samo jednim zajedničkim parametrom a to je „name“ od imena skupine levela(stage). Svaki level unutar ove Json datoteke ima svoje ime,veličina,pomoćne varijable, početnu matricu i matricu „solution“ koja predstavlja rješenje zagonetke.

5.ZAKLJUČAK

Tema ovog završnog rada je razvoj logičke igre stabla i šatori koristeći Java programerski jezik uz LibGDX softverski okvir i softversko okruženje Android Studio. LibGDX Aplikacija je uspješno napravljena koristeći mnoge druge tehnologije uključujući: JSON Format, XML Format, GIMP itd. Prilikom razvoja igre mogli su se vidjeti nedostatci LibGDX softverskog okvira prilikom skaliranja veličine grafičkih elemenata i postavljanjem na ispravnu lokaciju. U ovome završnom radu se prošlo kroz cijeli postupak razvoja igre. Na početku se moglo upoznati sa svim tehnologijama koje potrebne tehnologije za razvoj aplikacije, konstrukcije grafičkih elemenata koji će se koristiti unutar aplikacije, programerskog i softversko arhitektualnog dijela. Za razvoj ovakve igre se moglo koristiti drugačije razvojno okruženje poput Unity-a s kojim bi dobili slične rezultate. Neke od prednosti LibGDX su: Apache licenca dok Unity koristi komercijalnu licencu za poznatije aplikacije i puno bolji i fluidniji razvoj 2d igri. Za ovu igru se mogu kreirati novi dodatci koji bi povećali razinu zabave prilikom igranja.

LITERATURA

[1] Java The complete Reference Ninth Edition by Herbert Schildt, McGraw-Hill Education travanj 2014.

[2] Data Structures and Algorithms by Granville Barnet, Addison-Wesley Longman 2008.

[3] https://developer.android.com, zadnji pristup rujan 2018.

[4] https://libgdx.badlogicgames.com/documentation/, zadnji pristup rujan 2018.

[5] https://libgdx.badlogicgames.com/ci/nightlies/docs/api/com/badlogic/gdx, zadnji pristup rujan 2018.

SAŽETAK

U ovom završnom radu je se radilo na razvoju igre stabla i šatori koja je razvijena za više platformi u isto vrijeme koristeći LibGDX softverski okvira. Viđene su tehnologije koje su se koristile i njihove mogućnosti. Java se koristila kao programerski jezik za razvoj aplikacije. Android studio je služio kao softversko okruženje u kojem je se kod pisao i kompajlirao. Za spremanje podataka o levelima , koristio se JSON format. Uz podatke o levelima spremalo je se i ime skupine levela(stage). XML format se koristio za definiranje konfiguracijskih parametara poput imena aplikacije, lokacije ikone, teme koja se koristi itd. Nakon toga prošlo je se kroz pravile ove igre te ideju preko koje je ova igra realizirana. Zatim se mogao vidjeti primjer gdje je se rješila jedna zagonetka teđžine 5x5 blokova.Zatim se prikazao razvoj grafike korištene za razvoj igre i arhitektura koda. Pri prolasku kroz arhitekturu je se vidjeo abstraktan način prilasku dizajniranja koda radi veće upotrijebljivosti. Najvažniji detalji svake klase su se mogli vidjeti te je njihova uloga bila detaljno opisana. JSON Format je bio korišten za spremanje podataka bitnih za generiranje levela.

Ključne riječi:

Softversko okruženje, softverski okvir, UNICODE, XML, JSON, LibGDX, Android Studio, GIMP, kompajler, sprite.

ABSTRACT

In this project we could see development of Trees and Tents game. This game was developed for multiple platforms at the same time using LibGDX Framework. The technologies that were used and their capabilities were also observed. Java was used as a programming language for application development. The Android studio served as a software environment in which the code was written and compiled. To save the level data, JSON format was used. With the level information, the name of the stage group was also saved. XML format was used to define configuration parameters such as application name, icon location, theme being used etc. After that there was an example where a puzzle of 5x5 blocks was solved. Then the rules of this game were explained and the idea through which this game was realized. Then there was introduction to the development of graphics used and code architecture. While working on architecture, the abstract way of designing the code was introduced for greater usability. The most important details of each class could be seen and their role was described in detail. JSON Format has been used to store data that is important for generating levels.

Keywords:

IDE, Framework, UNICODE, XML, JSON, LibGDX, Android Studio, GIMP, compiler, sprite.

ŽIVOTOPIS

Autor ovog završnog rada je Igor Kukolj, rođen 26. lipnja 1996. g. u Žepču, Bosna i Hercegovina. Otac mu je Ilija Kukolj a majka Jela Kukolj. 1997 g. se seli u Tovarnik, Hrvatska. Sa 7 godina upisuje Osnovnu školu Antun Gustav Matoš Tovarnik. Poslije završetka osnovne škole upisuje Tehničku školu Ruđera Boškovića Vinkovci smjer Arhitektonski tehničar. Tijekom srednje škole programiranje ga je krenulo zanimati pa je naučio Javu i C# te je radio jednostavne Flash igre. Zbog novog hobija odlučuje promjeniti granu sa građevine na elektrotehniku te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek smjer Informatika.

Potpis: _____

PRILOZI (DVD)

Prilog 1. Pisana verzija završnog rada .docx formatu

Prilog 2. Pisana verzija završnog rada u .pdf formatu

Prilog 3. Kompletan kod aplikacije