Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Tušar

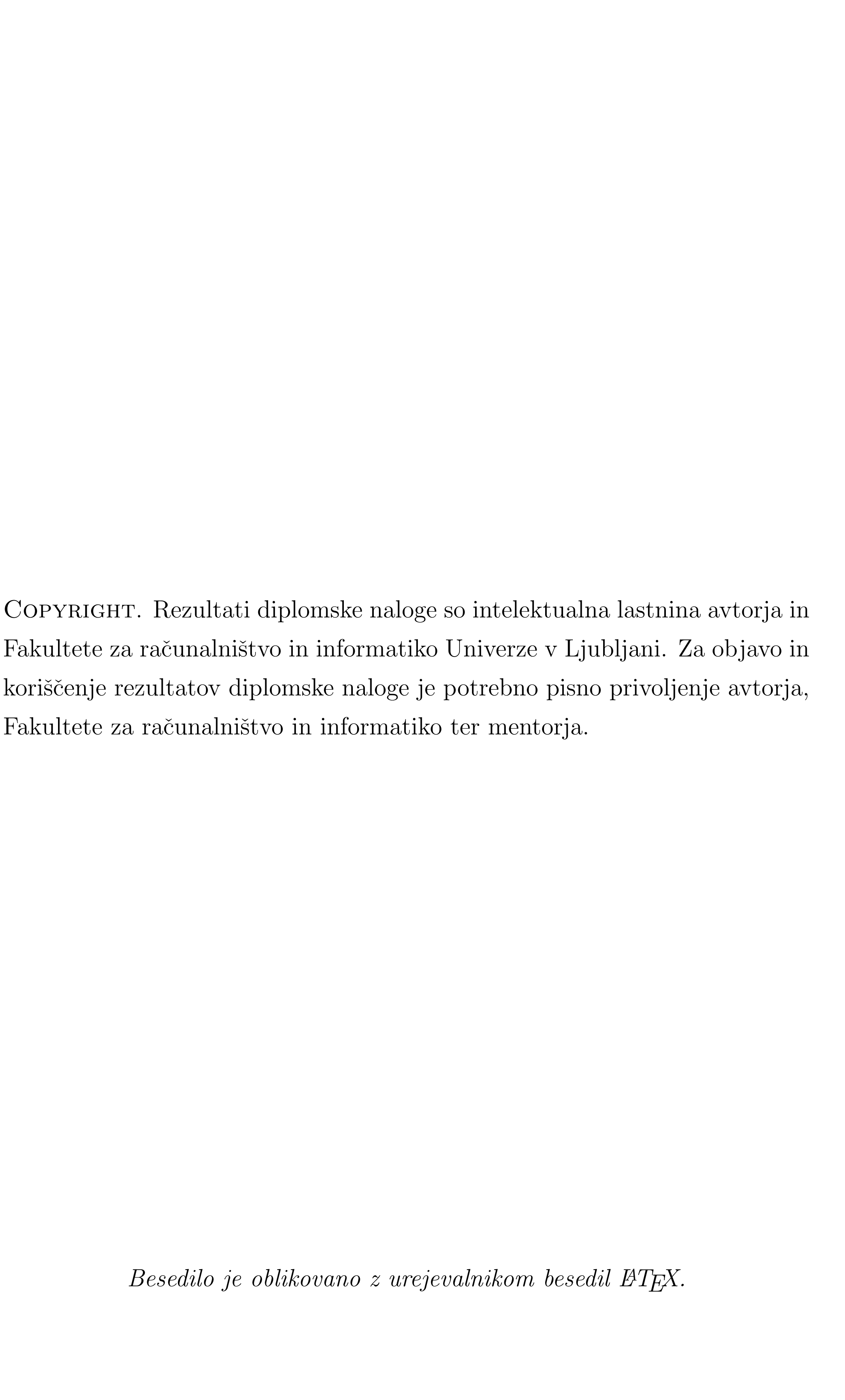
Generiranje programske kode kolutnih iger

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2019



Copyright. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Generiranje programske kode kolutnih iger

Tematika naloge:

Aplikacije iz specifične domene imajo pogosto podobne značilnosti in strukturo, kar omogoča avtomatizacijo njihove izdelave. Avtomatsko generiranje programske kode lahko prihrani veliko časa in prispeva k boljši kakovosti rešitve, v kolikor ga je mogoče ustrezno izvesti.

V diplomski nalogi predstavite pristop za generiranje programske kode za kolutne igre. V ta namen kratko opišite avtomatsko generiranje programske kode ter posebnosti kolutnih iger. Na podlagi teh predlagajte ustrezen postopek in naredite generator, ki omogoča tvorbo programske kode za osnovne funkcionalnosti tovrstnih iger. Rešitev preverite na več primerih in jo ocenite.

Kazalo

Povzetek

Abstract

1 Uvod 1

2 Avtomatsko generiranje programske kode 3

2.1 Avtomatsko programiranje 3

2.2 Generiranje izvorne kode 5

2.2.1 Program za izdelavo predloge 7

2.2.2 Pretvornik 7

3 Kolutne igre 9

3.1 Osnovni pojmi pri kolutnih igrah 10

3.2 Delovanje kolutne igre 11

3.3 Vrste dobitkov 13

3.3.1 Dobitek na plačilni liniji 13

3.3.2 Dobitki na kolutih 13

3.3.3 Dodatni dobitki 15

3.4 Podrobnosti posameznih modulov 17

3.4.1 Osnovni modul 17

3.4.2 Brezplačni igralni modul 18

3.4.3 Dodatna igra 18

4 Pregled uporabljenih tehnologij in orodij **21**
4.1 Pregled uporabljenih tehnologij 21
4.1.1 Java 21
4.1.2 Knjižnica `minimal-json` 22
4.1.3 Orodje Gradle 22
4.1.4 Podatkovni format JSON 23
4.2 Uporabljena programska orodja 24
4.2.1 IntelliJ IDEA 24
4.2.2 Notepad++ 24

5 Izvedba generatorja **25**
5.1 Idejna rešitev 25
5.1.1 Podatki v konfiguraciji 26
5.1.2 Delovanje generatorja 27
5.2 Razvoj generatorja 28
5.2.1 Pridobivanje podatkov 28
5.2.2 Zapisovanje podatkov 34
5.3 Preverjanje delovanja 35

6 Analiza **39**
6.1 Analiza pospešitve izdelave igre 39
6.1.1 Mammoth chase 40
6.1.2 Gates of Babylon 41
6.1.3 Težave analize 44
6.2 Težave z generatorjem 44
6.3 Ugotovitev 45

7 Zaključek **47**

Literatura **50**

Seznam uporabljenih kratic

kratica	angleško	slovensko
kratica	**angleško**	**slovensko**
LCDP	Low-Code Development Platform	platforma za razvoj nizkonivojske kode
DSL	Domain Specific Language	domensko specifični jeziki
RTP	Return To Player	odstotek povrnjenega denarja igralcu
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
XML	eXtensible Markup Language	razširljiv označevalni jezik
JSON	JavaScript Object Notation	notacijski Javascript objekt
IDE	Integrated Development Environment	integrirano razvojno okolje

Povzetek

Naslov: Generiranje programske kode kolutnih iger

Avtor: Simon Tušar

Diplomsko delo obravnava generiranje programske kode za domeno izdelave kolutnih iger. Generator programske kode je program, ki iz ustreznih vhodnih podatkov tvori programsko kodo.

Kolutne igre so najbolj razširjena vrsta iger na srečo, ki simulira klasične igralne avtomate. Priljubljene so postale zaradi svoje enostavnosti, saj igralec le stavi izbrano količino denarja, igra pa se izvede s pritiskom na gumb.

Tovrstne igre imajo zelo podobno strukturo, zato so primerne za avtomatsko generiranje programske kode zanje. S tem pristopom želimo zmanjšati čas, ki ga porabimo za njihovo ročno izdelavo. V nalogi so prikazane tehnične podrobnosti izdelave generatorja programske kode za kolutne igre, ki omogoča tvorbo programske kode za osrednji del igre.

Analiza po implementaciji je pokazala, da smo z generatorjem prihranili kar nekaj časa, zmanjšalo pa se je tudi število napak pri izdelavi.

Ključne besede: generiranje programske kode, igralni avtomat, kolutne igre, JSON.

Abstract

Title: The generation of the source code for slot games

Author: Simon Tušar

This thesis addresses the problem of source code generation for slot games. Source code generator is a computer program that generates source code from the provided configuration data.

Slot games are amongst the most popular gambling games. They simulate the classic slot machines. They became popular because they are easy to play; the player simply chooses the amount of money to bet and starts the game with a press on the button.

As such games have very similar structure we can automaticaly generate source code for them. With this approach we want to reduce the time needed for their implementation. This thesis describes the technical details of the developed source code generator for slot games.

Our analysis has shown that the developed code generator saves a lot of development time and reduces the number of mistakes that are made during the manual development.

Keywords: source code generation, parser, slot games, JSON.

Poglavje 1

Uvod

Nekateri programi so si tako podobni, da se razlikujejo le po svoji konfiguraciji. Konfiguracija nam v tem primeru predstavlja posamezne oblikovne elemente, ki ne vplivajo na način delovanja programa. Pri določenih spletnih straneh je konfiguracija na primer slika z logotipom podjetja, pri čemer je spletna stran vedno sestavljena iz enakih elementov, ki se izpišejo v istem zaporedju (na primer osnovni podatki o podjetju). V primeru enostavne konfiguracije so programi tako, z izjemo skromnih sprememb, praktično enaki. Njihovo izdelavo bi pospešili tako, da bi omogočili vstavljanje spremenljivih parametrov na določena mesta v vnaprej določeno predlogo. Ko je konfiguracija zahtevnejša in je potrebno nastaviti veliko različnih parametrov, bi bilo dobro narediti generatorski program, ki bi nam na podlagi začetne konfiguracije avtomatsko tvoril relativno zapleteno programsko kodo.

Z generatorji lahko rešujemo različne probleme. Ena izmed večjih tovrstnih problemskih domen je izdelava spletnih strani, saj le-te pogosto funkcionirajo podobno, najbolj opazna razlika pa je njihov izgled. S pomočjo generatorja kode bi lahko pospešili tudi izdelavo programske kode za igralne avtomate. Tudi delovanje slednjih je zelo podobno. Med seboj se večinoma razlikujejo le po izgledu igralne površine. Podoben problem predstavlja tudi izdelava uporabniških vmesnikov. Tu imajo posamezna polja podobne zahteve (na primer gesla se ne smejo prikazati), ki bi jih lahko s pomočjo konfi-

guracije preprosto nastavili. Poleg teh praktičnih primerov obstaja še veliko drugih, pri katerih bi razvijalcu generator kode precej olajšal delo.

V diplomski nalogi bomo najprej opisali problemsko domeno avtomatskega generiranja izdelave programske kode. Temu poglavju bo sledilo poglavje, ki bo podrobneje predstavilo konkretno problemsko domeno kolutnih iger. Sledilo bo poglavje o uporabljenih tehnologijah in orodjih, nato pa še eno o poteku implementacije generatorja kode zanje in nekaj podrobnosti, ki smo jih ob tem srečali. Za konec bomo naredili še analizo, s katero bomo potrdili uspešnost opravljenega dela.

Poglavje 2

Avtomatsko generiranje programske kode

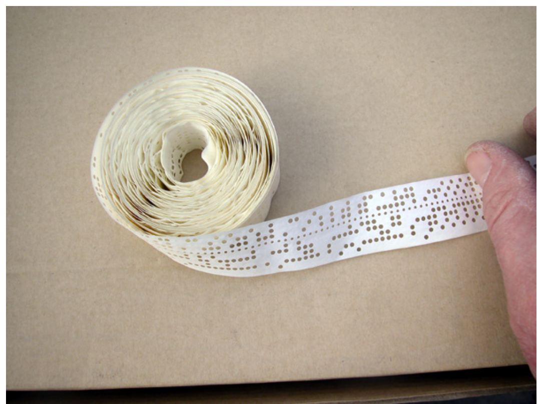
Naloga se nanaša na avtomatizirano generiranje programske kode. To področje je znano že dolgo, obravnavamo pa ga lahko različno. Naš cilj je narediti programsko orodje, ki bo na podlagi definirane konfiguracije tvorilo programsko kodo, ki bo čim bližje končni obliki. V ta namen želimo predstaviti ustrezne teoretične pristope, ki nam bodo pri tem v pomoč.

2.1 Avtomatsko programiranje

Avtomatsko programiranje (ang. automatic programming) opisuje način programiranja, pri katerem s pomočjo posebnega programa generiramo novo programsko kodo glede na vnaprej določene specifikacije [4]. V okviru tega poznamo različne možnosti, vendar v splošnem takšni programi programerju omogočajo kodo pisati preprosteje in hitreje [5].

Definicija avtomatskega programiranja se je skozi čas spreminjala. V štiridesetih letih prejšnjega stoletja je avtomatsko programiranje opisovalo avtomatizacijo izdelave luknjastih trakov. Luknjast trak, ki je prikazan na sliki 2.1, je namenjen shranjevanju podatkov [24]. Kasneje se je ta pojem nanašal na prevajanje visokonivojskih programskih jezikov (recimo ALGOL

in Fortran) v strojni jezik. Takšnim programom sedaj pravimo prevajalniki (ang. compiler) [5]. Danes je pojem avtomatskega programiranja najbližje generiranju zahtevne programske kode iz čim preprostejšega vhoda.



Slika 2.1: Slika luknjastega traku [25]

Programi, ki so namenjeni avtomatskemu programiranju, skrajšajo čas izdelave določenega programa, poleg tega pa programerju prihranijo vložen trud, ki bi ga moral porabiti za izdelavo (rutinskega dela) programa. Običajna alternativa avtomatskemu programiranju je ročna implementacija. Tudi če izvzamemo potreben čas za implementacijo ta alternativa ni najboljša, saj je potrebno ob kasnejših spremembah dokumentacije kodo spet ročno popraviti, poleg tega pa z ročnim pristopom lažje naredimo napake [8].

Poznamo različne vrste avtomatskega programiranja [5]:

- Generativno programiranje (ang. generative programming) predstavlja koncept avtomatizirane izdelave programskih komponent s ponovno uporabo programske kode (ang. reusability).

- Generiranje izvorne kode (ang. source-code generation) predstavlja proces generiranja izvorne kode glede na opis problema ali ontološkega modela (ang. ontological model).

- Izdelava s pomočjo platforme za razvoj nizkonivojske kode (ang. low-code development platform) predstavlja rešitev, pri kateri razvijalec

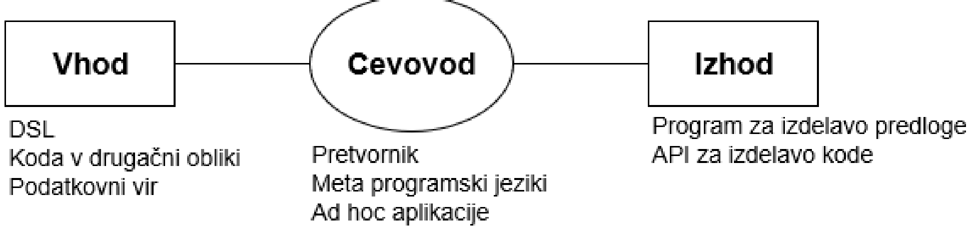
kodo ustvari s pomočjo grafičnega programskega vmesnika in konfiguracije.

Izmed teh si bomo za podrobnejši pregled izbrali proces generiranja izvorne kode.

2.2 Generiranje izvorne kode

Proces generiranja programske kode izhaja iz opisa problema ali ontološkega modela. Generiranje kode je doseženo s programskimi orodji (kot so na primer programi za izdelavo predlog), lahko pa tudi s pomočjo integriranih razvojnih orodij (IDE). Ta orodja omogočajo, da programsko kodo tvorimo na različne načine, ki so učinkovitejši kot ročno programiranje. [5]. Proces generiranja programske kode tudi lahko služi zgolj za izboljšanje učinkovitosti ali kot ključen del procesa izdelave programa [7].

Obstaja več izvedb generatorjev programske kode. V osnovni izvedbi, ki je prikazana na sliki 2.2, ti potrebujejo **vhod**, **izhod** ter **cevovod**, preko katerega bo generator iz vhodnih parametrov naredil željen izhod [7].



Slika 2.2: Prikaz elementov osnovnega generatorja izvorne kode [7]

Pri teh velja:

- Vhod obsega parametre, ki so potrebni za generiranje kode,
- izhod opisuje, v kakšni obliki bomo dobili programsko kodo.

Možni vhodni parametri programa za generiranje izvorne kode so:

- Domensko specifični programski jezik (ang. domain-specific language) je poseben jezik, s katerim z uporabo enostavnega jezika, ki je uporabniku bolj razumljiv, opišemo vsebino na preprostejši način.

- Koda v drugačni obliki je zapis vsebine v neprogramski obliki (na primer sheme podatkovne baze), iz katere lahko generiramo programsko kodo.

- Podatkovni vir opisuje drugačen način (ali format) za opis vhodnega vira (Excel, CSV, podatkovna baza), iz katerega razberemo ustrezne podatke za tvorbo programske kode.

Z danimi vhodnimi parametri generiramo programsko kodo preko cevovoda. Tu so najbolj znani cevovodi za generiranje kode:

- Pretvornik (ang. parser), ki iz datoteke prebere podatke in jih prepiše v ustrezno predlogo programske kode.

- Namenske aplikacije (ang. ad hoc applications) so aplikacije, ki poleg vhoda upoštevajo še druge podatke in tako tvorijo izhod.

- Generatorji v razvojnih okoljih (krajše IDE) nam omogočajo tvorbo osnovnih metod razredov (`toString`, `hashCode`, `equals` ...) po vnaprej določenih predlogah.

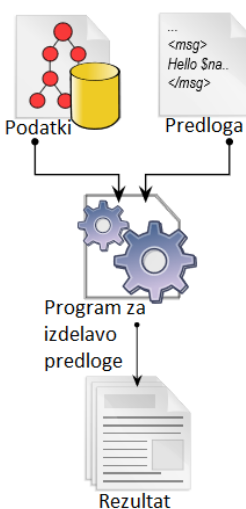
Pri programih za generiranje programske kode imamo dva možna izhoda:

- Program za izdelavo predloge (ang. template engine) nam omogoča zgolj zapolnitev določenih podatkov v dani predlogi.

- Aplikacijski programski vmesnik za izdelavo kode (ang. code building APIs) nam omogoča programsko tvorbo datotek, ki jih zna prevajalnik uspešno prevesti v strojni jezik.

2.2.1 Program za izdelavo predloge

Program za izdelavo predloge (ang. template engine, template processor) je program, ki združi predlogo s podatki, da naredi željeno datoteko s programsko kodo [30]. Shemo vidimo na sliki 2.3.



Slika 2.3: Osnovni elementi programov za izdelavo predlog [9]

Tovrstni programi vključujejo različne funkcionalnosti s poudarkom na procesiranju besedila. Programi za izdelavo predlog lahko tvorijo različne vrste izhodov. Ti so bodisi dokumenti, spletne strani ali programska koda.

2.2.2 Pretvornik

Pretvarjanje (ang. parsing) je proces analize simbolov ali besed [28]. V računalništvu se pojem pretvarjanje nanaša na sintaktično analizo vhodne kode z namenom pisanja prevajalnikov (ang. compiler) in tolmačev (ang. interpreter). Poleg tega se lahko nanaša tudi na delitev ali ločitev besedila.

Pretvornik je programska komponenta, ki vhodni parameter (ponavadi je to besedilo) pretvori v podatkovno strukturo. Vhodni parameter se po navadi zapiše v obliki programskega jezika, lahko pa je tudi v obliki naravnega jezika ali tekstovnih podatkov. Glede na vrsto vhodnega parametra tako poznamo:

- Podatkovne jezike, kjer je glavna naloga pretvornika branje podatkov iz datotek, kot so na primer HTML, XML in podobne.
- Programske jezike, kjer je pretvornik del prevajalnika ali tolmača, ki pretvarja višjenivojsko programsko kodo v nižjenivojsko programsko kodo.

Pretvorniki pogosto dopolnjujejo programe za izdelavo predlog [28]. Orodja, ki uporabljajo pretvornike za generiranje programske kode, so zelo uspešna, saj so le-ti že obsežno raziskani, zaradi česar rešitve, ki omogočajo pretvarjanje v več različnih jezikov, že obstajajo [7].

Poglavje 3

Kolutne igre

Kolutne igre (ang. slot games) so zelo popularne računalniške igre na srečo [26]. Dotične igre so se razvile iz klasičnih igralnih avtomatov [33]. Njihova največja prednost je enostavnost igranja, zaradi česa lastnik z njimi običajno zasluži več denarja kot z zahtevnejšimi igrami [32]. Izgled preproste kolutne igre je prikazan na sliki 3.1.

Slika 3.1: Prikaz osnovnih pojmov v kolutni igri [13]

3.1 Osnovni pojmi pri kolutnih igrah

Za lažje razumevanje bomo najprej našteli in opisali nekaj ključnih pojmov, ki jih bomo uporabljali v nadaljevanju diplomskega dela. Nekateri osnovni pojmi so prikazani na sliki 3.1. Osnovni pojmi, ki jih bomo uporabili pri opisu kolutnih iger so:

- **Simbol** (ang. symbol) je prikazan simbol, ki se nahaja na kolutih igre.
- **Kolut** (ang. reel) je stolpec, ki ga sestavlja več simbolov.
- **Prikazani koluti** (ang. slot face) so skupina več kolutov, ki so prikazani igralcu. Prikazani so na sliki 3.1.
- **Plačilna linija** (ang. payline) je določeno zaporedje simbolov na kolutih. V primeru na sliki 3.1 je prikazana plačilna linija številka štiri, ki na prvem kolutu vzame prvi simbol od zgoraj, na drugem kolutu drugi simbol, na tretjem kolutu tretji simbol, na četrtem kolutu vzame drugi simbol in na petem kolutu prvi simbol. Ta plačilna linija se nato preverja za možne dobitke. V primeru s slike 3.1 je ta dobitna, ker trije simboli od petih predstavljajo dobitno kombinacijo.
- **Podatki o stavi** igralcu povejo, kako in koliko denarja je stavil. Tu moramo omeniti več različnih pojmov:
  - **Število plačilnih linij** (ang. number of paylines) označuje, za kolikšno število plačilnih linij želi igralec igrati s trenutno stavo.
  - **Velikost kovanca** (ang. coin size) označuje vrednost kovanca.
  - **Število kovancev** (ang. coins) pove, koliko kovancev bo igralec stavil na plačilno linijo.
  - **Velikost stave** (ang. bet) določa skupni znesek trenutne stave.
  - **Stava na plačilno linijo** (ang. line bet) pove, kolikšen je vplačan znesek za eno plačilno linijo.

- **Simbol za zamenjavo** (ang. wild) je simbol, ki lahko na plačilni liniji nadomesti katerikoli simbol.

- **Dobitek na igralnih kolutih** (ang. scatter), je zadetek, ki ga igralec zadane, če se na igralcu prikazanih kolutih večkrat pojavi simbol, ki izplačuje ta dobitek. Pri tem ni važno, kje se ta simbol pojavi.

- **Plačilna lista** (ang. pay table) nam pove, kakšni so dobitki oziroma koliko simbolov je potrebno zbrati za določen dobitek.

- **Način plačevanja čez vse možne plačilne linije** (ang. ways pay) je način preverjanja simbolov na kolutih, pri čemer preverjamo vse možne plačilne linije. Pri igri, ki ima pet kolutov in na vsakem od teh tri simbole, je število možnih plačilnih linij 243 (3^5).

- **Modul** (ang. modul) je ime, ki se uporablja pri delitvi igre na več različnih delov. Igre so v večini primerov sestavljene iz dveh modulov, ki se lahko razlikujeta po logiki in/ali konfiguraciji.

- **RTP** (ang. Return To Player) predstavlja odstotek denarja, ki bi ga igralec prejel nazaj, če to igro igra neskončnokrat.

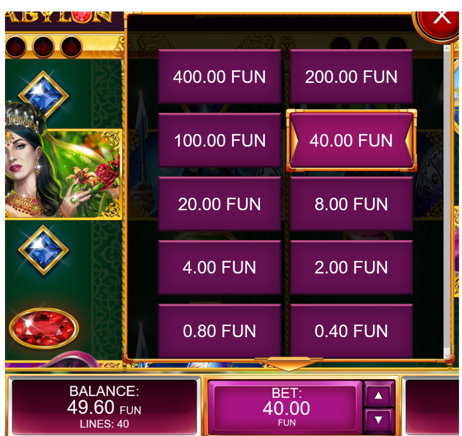
3.2 Delovanje kolutne igre

Igralec sproži igro s pritiskom na gumb za igranje. Pred tem izbere količino denarja, ki ga želi staviti na način, ki je je prikazan na sliki 3.2.

Ko je stava vnesena in igralec pritisne na gumb, se začne izvajati programska koda, ki simulira vrtenje kolutov. Tu se najprej določijo končne pozicije igralnih kolutov. Pozicije kolutov povedo, kateri simboli se pojavijo na vseh kolutih.

Naslednji korak je preverjanje dobitkov. Slednji se delijo na več vrst:

- **Dobitek na plačilni liniji** se zgodi takrat, ko se na plačilni liniji zaporedoma nahajajo enaki simboli.



Slika 3.2: Izbira stave pri igri Gates of Babylon [12]

- **Dobitek na kolutih** (ang. scatter) pomeni, da je število določenih simbolov na vseh kolutih večje ali enako številu simbolov, ki ga določa konfiguracija igre.

- **Dodatni dobitki** so vsi ostali dobitki.

Po obravnavi dobitkov se igra ponavadi zaključi, razen v primerih, če igralec zadane brezplačne vrtljaje ali dodatno igro.

Pri igranju kolutnih iger poznamo različne module, ki jih lahko vsebuje igra. Moduli so lahko:

- **Osnovni modul** (ang. slot modul) je običajni način igre, ki ga igralec igra, ko vplača stavo. Ta modul je prikazan na sliki 3.1.

- **Modul brezplačne igre** (ang. free spin) je način igre, ki ga igralec lahko zadane. Tu igralcu za igranje ni potrebno ponovno vplačati stave za ponovno igranje, sicer pa se igra enako kot osnovni modul.

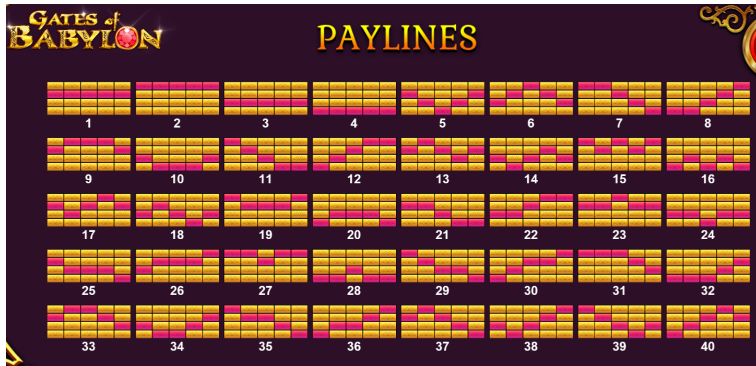
- **Dodatni modul** je modul, ki ne vsebuje kolutov, ampak simulira dobitke. Pri tem se dobitek naključno izbere izmed vseh možnih dobitkov.

3.3 Vrste dobitkov

V tem podpoglavju bomo na kratko predstavili različne vrste dobitkov.

3.3.1 Dobitek na plačilni liniji

Dobitek na plačilni liniji je vrsta dobitka, pri katerem je potrebno, da je na plačilni liniji zapovrstjo določeno število enakih simbolov. Potek možnih plačilnih linij je prikazan na sliki 3.3.



Slika 3.3: Možne plačilne linije pri igri Gates of Babylon [12]

Dobitek na plačilni liniji se izplača tako, da se dobitek za določeno število simbolov pomnoži z velikostjo stave, ki je bila vplačana na eno plačilno linijo. V primeru s slike 3.2 smo stavili 40 FUN [1] za 40 plačilnih linij. To pomeni, da smo na eno plačilno linijo stavili 1 FUN. Faktor za tri enake simbole princes, ki smo jih zadeli na plačilni liniji s slike 3.4, je 15. Lahko ga razberemo s plačilne tabele na sliki 3.5. Skupni zadetek je tako 15 FUN.

3.3.2 Dobitki na kolutih

Dobitki na kolutih (ang. scatter) so dobitki, pri katerih mora biti na vseh kolutih skupaj določeno število simbolov. Ti dobitki nam ponavadi za nagrado prinašajo dodatno igro (večinoma modul z brezplačnim igranjem ali dodatno igro). Poleg tega se od navadnih dobitkov na plačilnih linijah razlikujejo v

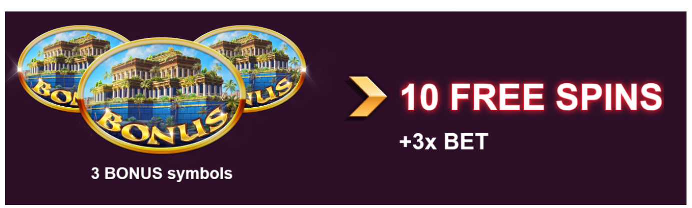
[1] FUN je izmišljena denarna enota, ki se uporablja pri igranju te igre

Slika 3.4: Zadeta plačilna linija pri igri Gates of Babylon [12]

Slika 3.5: Del plačilne tabele pri igri Gates of Babylon [12]

tem, da se dobitki pomnožijo s celotno zadnjo stavo. V primeru s slike 3.2 se 40 FUN pomnoži z dobitkom iz plačilne tabele (na sliki3.7). Faktor za tri simbole BONUS je 3. Igralec tako dobi 120 FUN, dobi pa tudi 10 brezplačnih vrtljajev. Dobitek na kolutu je prikazan na sliki 3.6.

Slika 3.6: Simbol, imenovan Bonus, prinaša dobitek na kolutu pri igri Gates of Babylon. [12]



Slika 3.7: Del plačilne tabele pri igri Gates of Babylon [12]

3.3.3 Dodatni dobitki

Dodatni dobitki se izplačujejo poleg dobitkov, ki jih zadenemo na kolutih. Takšni dobitki so:

- **zbiranje simbolov na kolutih v enem vrtljaju**,
- **misije** in
- **zbiranje simbolov za dodatno igro**.

Zbiranje simbolov na kolutih v enem vrtljaju deluje podobno kot dobitki na kolutih, le da se tu za dobitke štejejo tudi zamenjalni simboli, poleg tega pa ti dobitki nikoli ne prinašajo dodatne igre ali brezplačnih vrtljajev. Omenjeni dobitki so sicer redki, saj mora biti na vseh kolutih skupaj prisotno veliko število določenih simbolov, zaradi česar je visoko tudi njihovo izplačilo.

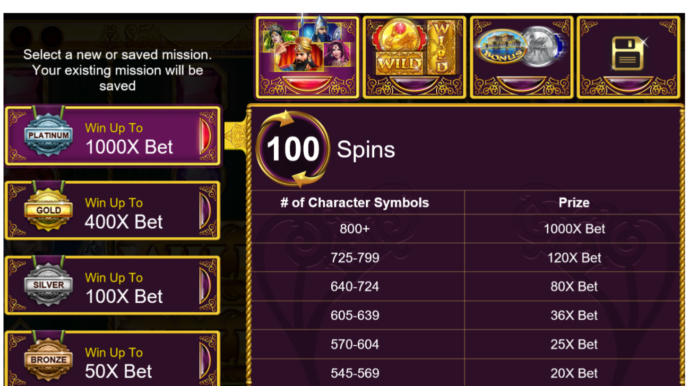
Zadetek tovrstnega dobitka je prikazan na sliki 3.9. Ker je število rdečih simbolov in zamenjalnih simbolov (z napisom "wild") na vseh kolutih skupaj večje kot 8, simboli z oznako "2x" štejejo dvakrat. Na sliki imamo tako 3-krat po enega rdečega tigra, 2-krat po dva rdeča tigra in en zamenjalni simbol, kar pomeni, da imamo skupaj 8 simbolov. Ti simboli skupaj nam izplačajo dobitek "Silver bonus jackpot".

Misije so dobitki, pri katerih v določenem številu vrtljajev nabiramo izbrane simbole. Če presežemo določeno število izbranih simbolov, se nam izplača dobitek glede na to, koliko simbolov smo zadeli v teh vrtljajih. Pri tem lahko izbiramo simbole za zbiranje in število vrtljajev, v katerih bomo zbirali te simbole.

Slika 3.8: Prikaz zadetka pri zbiranju simbolov na kolutih v enem vrtljaju pri igri Tiger claws [16]

Slika 3.9: Del plačilne tabele pri igri Tiger claws [16]

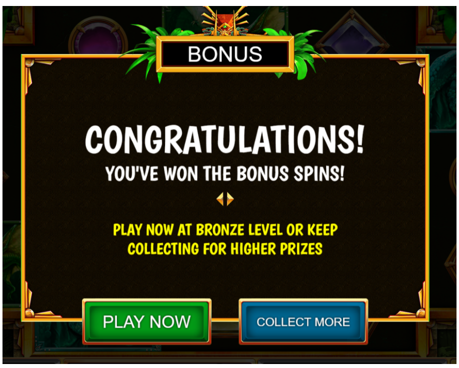
Slika 3.10 prikazuje izbiro misij. V primeru, izbrane označene misije, ko med igranjem le-te zberemo več kot 800 simbolov princev ali princes, bomo dobili dobitek, ki je 1000-krat večji od celotne stave. V primeru stave 40 FUN bi na koncu dobili kar 40000 FUN.



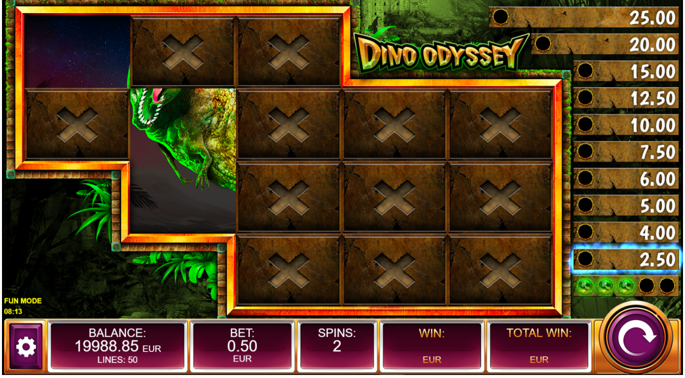
Slika 3.10: Slika misij igre Gates of Babylon [12]

Pri zbiranju simbolov za dodatno igro igralec zbira izbrane simbole. Ko preseže določeno število zbranih simbolov, lahko nadaljuje z zbiranjem ali

začne z igranjem dodatne igre, saj imajo določene igre več različnih stopenj. Če je igralec že na zadnji stopnji se dodatna igra začne sama. Okno je prikazano na sliki 3.11. Dodatna igra je prikazana na sliki 3.12.



Slika 3.11: Možnost za igranje dodatne igre pri igri Dinno odyssey [11]



Slika 3.12: Dodatne igre, ki jo lahko igralec zadane z zbiranjem simbolov pri igri Dinno odyssey [11]

3.4 Podrobnosti posameznih modulov

3.4.1 Osnovni modul

Osnovni modul (ang. slot) je modul, ki se igra vsakič, ko igralec s pritiskom na gumb zavrti kolute. Delovanje tega modula na zaledni strani poteka tako,

da najprej določimo končne pozicije kolutov. Iz tega lahko nato določimo, kateri simboli so prikazani uporabniku, nazadnje pa izračunamo dobitke.

Iz omenjenega modula lahko igralec preide v drug modul, če le-tega zadane. Ta modul, ki je prikazan na sliki 3.13, ima lahko tudi posebnosti. V primeru igre Gates of Babylon je ta posebnost zbiranje zamenjalnih simbolov (ang. wild) na določenih kolutih (slika 3.14). Ko se zberejo trije taki simboli, se vsi simboli na kolutu za naslednjih nekaj vrtljajev spremenijo v zamenjalne simbole, kar omogoča več dobitkov.

Slika 3.13: Osnovni modul pri igri Gates of Babylon [12]

3.4.2 Brezplačni igralni modul

Modul brezplačne igre (ang. free spin) je modul, v katerega igralec preide, ko v osnovnem modulu zadane brezplačni modul. V njem lahko igralec zadane še dodatne vrtljaje, ki so prav tako brezplačni. Brezplačni igralni modul sicer deluje enako kot osnovni modul (določi se končne pozicije kolutov, iz katerih se izpelje simbole za prikaz in preveri dobitke), lahko pa ima tudi svoje posebnosti. Tako slika 3.15 na primer prikazuje posebnost pri igri Gates of Babylon, ko je zadnji kolut zaklenjen, ker je igralec zbral določeno število zamenjalnih simbolov.

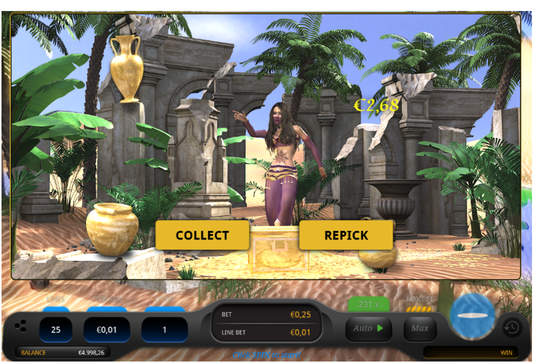
3.4.3 Dodatna igra

Dodatna igra je igra, ko se v ozadju ne vrtijo koluti, temveč je omogočena le izbira dobitka. Zaledni del igre izmed možnih dobitkov naključno izbere enega. Igralcu se to prikaže tako, da se mu ponudi več možnosti. Ob pritisku

Slika 3.14: Zadnji kolut vsebuje le zamenjalne simbole v igri Gates of Babylon [12]

Slika 3.15: Izgled brezplačnega modula igre Gates of Babylon [12]

na eno izmed le-teh igralec zadane določeno vsoto denarja. Dodatna igra pri igri Golden dunes je prikazana na sliki 3.16.



Slika 3.16: Dodatna igra pri igri Golden dunes [14]

Poglavje 4

Pregled uporabljenih tehnologij in orodij

V dotičnem delu bomo našteli in opisali tehnologije in orodja, ki smo jih uporabili pri izdelavi diplomske naloge. Poleg tega bomo navedli tudi razloge za uporabo teh tehnologij in orodij.

4.1 Pregled uporabljenih tehnologij

4.1.1 Java

Java je objektno usmerjen programski jeziki [6]. Ena izmed ključnih prednosti tega programskega jezika je, da enaka koda pravilno deluje na vseh platformah, ki jih programski jezik podpira [1].

Javo smo za izdelavo naše rešitve izbrali, ker je zelo priljubljen programski jezik. Na spletu zanjo obstaja veliko dokumentacije. Popularnost je razvidna tudi iz tabele 4.1. Popularnost posameznih programskih jezikov je določena iz števila izkušenih programerjev in števila internetnih izobraževanj za določen jezik.

	Programski jezik	Ocena
1	Java	16.90%
2	C	13.34%
3	Python	8.29%
4	C++	8.16%
5	Visual Basic .NET	6.46%

Tabela 4.1: TIOBE Index popularnosti programskih jezikov [31]

4.1.2 Knjižnica `minimal-json`

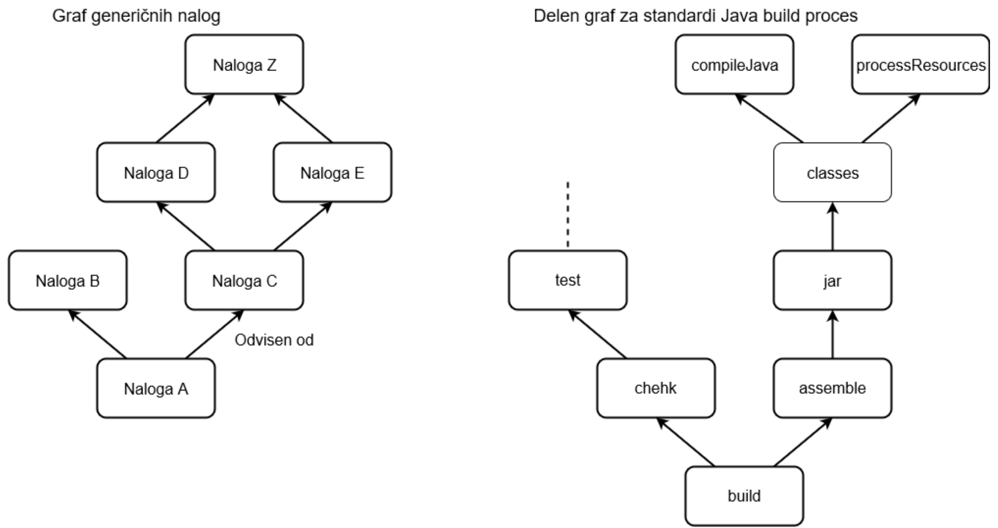
Java sama po sebi ne vsebuje pretvornika za datoteke tipa JSON, zato smo uporabili knjižnico, s katero bomo lahko besedilo prebrali in pretvorili v objekt JSON. Ta knjižnica je `minimal-json` [23]. Omogoča nam, da iz sintaktično pravilnega besedila naredimo objekt z ustrezno vsebino, ki ga lahko nato obdelujemo. To knjižnico smo izbrali, ker smo jo v preteklosti že uporabili in poznamo njene osnovne funkcionalnosti.

4.1.3 Orodje Gradle

Za dodajanje knjižnice v projekt smo uporabil Gradle [10]. To je odprtokodno programsko orodje, ki skrbi za avtomatizacijo gradnje projektov [21]. Dograjuje koncepte Apache Anti [3] in Apache Maveni [2].

Osnovni model tega orodja temelji na nalogah, ki jih povežemo z ustreznimi odvisnostimi. Primer je prikazan na sliki 4.1.

Primer prikazuje generičen graf nalog. Pred nalogo A je potrebno narediti nalogi B in C. Nalogo B lahko naredimo takoj, saj ni odvisna od nobene druge naloge. Naloga C pa je odvisna od nalog D in E, kar pomeni, da moramo prej opraviti nalogi D in E. Ti nalogi sta odvisni od naloge Z, zato je najprej potrebno rešiti slednjo. Po izvedbi naloge Z izvedemo nalogi D in E, sledi še naloga C, ki nam (po razrešitvi naloge B) omogoči, da naredimo nalogo A. Enak koncept velja za aktivnosti pri izvedbi javanskega projekta, ki je tudi prikazan na sliki 4.1.



Slika 4.1: Izgled grafa za izgradnjo projekta [21]

Za uporabo tega orodja smo se odločili, ker se le-to pogosto uporablja pri razvoju večjih javanskih aplikacij.

4.1.4 Podatkovni format JSON

JSON (ang. JavaScript Object Notation) je format za izmenjavo podatkov, ki je razumljiv tako človeku kot računalniku [19] . Zgrajen je na dveh splošnih podatkovnih strukturah:

- zbirki imen z vrednostmi ter
- urejenem seznamu teh vrednosti.

Primer JSON objekta vidimo na sliki 4.2. Za uporabo dotičnega formata za izmenjavo podatkov smo se odločili, ker v takšni obliki dobimoizhodiščno datoteko, iz katere bomo brali podatke o kolutni igri.

```
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
            "title": "S",
                        "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                    },
                                        "GlossSee": "markup"
                }
            }
        }
    }
}
```

Slika 4.2: Primer JSON objekta [20]

4.2 Uporabljena programska orodja

4.2.1 IntelliJ IDEA

IntelliJ IDEA je programsko orodje, ki je namenjeno razvijanju javanskih aplikacij. Ponaša se z veliko dodatnimi funkcionalnostmi, ki nam olajšajo izdelavo programa, med katerimi je tudi podpora za Gradle [18].

4.2.2 Notepad++

Za pregled datotek tipa JSON smo uporabili programsko orodje Notepad++ [27]. Program sam po sebi nima podpore za datoteke tipa JSON, zato smo mu dodali vtičnik, ki omogoča urejen pregled datotek [29]. Dotični vtičnik nam omogoča, da neurejene objekte JSON uredimo brez dodatnega truda.

Poglavje 5

Izvedba generatorja

V poglavju bomo predstavili potek implementacije generatorskega programa, ki nam poenostavi izdelavo kolutnih iger. Pri tem bomo najprej opisali idejo rešitve, sledil pa bo opis izdelave pretvornika, ki pretvori konfiguracijsko datoteko tipa JSON v programsko kodo igre.

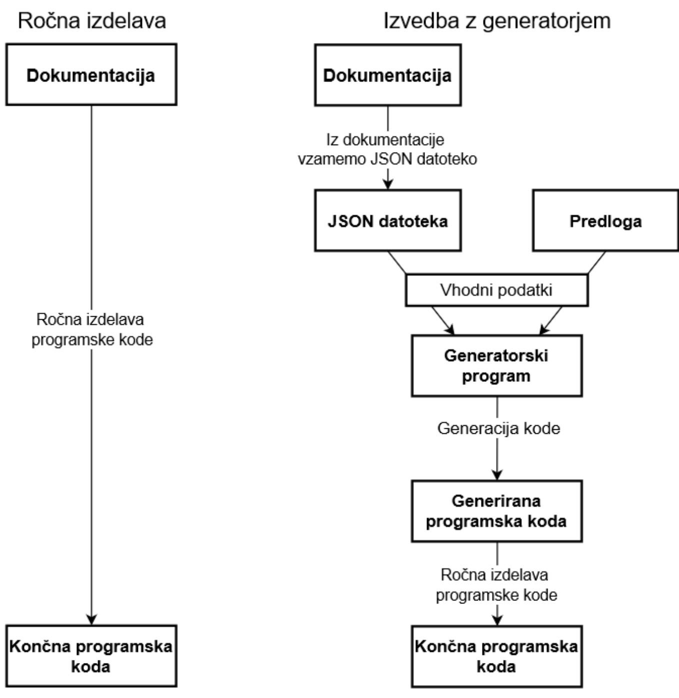
5.1 Idejna rešitev

Povedano splošno, je ideja napisati program, ki nam bo iz dane konfiguracije avtomatsko generiral programsko kodo kolutne igre. Slika 5.1 prikazuje, kako bo delovala naša rešitev.

Pri ročni izdelavi smo celotno programsko kodo napisali ročno na podlagi začetnega opisa konfiguracije v dokumentaciji. Kot je bilo že omenjeno, tovrstna izdelava zahteva veliko časa, zaradi ročnega pisanja kode pa se pojavljajo tudi težave z napakami.

Pri izvedbi z generatorjem bomo iz prejete dokumentacije izbrali datoteko v formatu JSON, ki vsebuje konfiguracijo igre. Za tipičen zgled datoteke bomo vzeli obstoječo datoteko, ki jo dobimo med dokumentacijo enega izmed ponudnikov[1], za katerega pogosto izdelujemo igre. Iz te datoteke bo generatorski program na podlagi vnaprej definirane predloge sestavil programsko

[1]Kalamba games [22]



Slika 5.1: Primerjava ročne izdelave in generatorja

kodo za osnovne funkcionalnosti. Na koncu bomo ročno dodali še dodatne funkcionalnosti, ki jih avtomatsko zelo težko generiramo. Podrobnosti so razvidne iz sheme s slike 5.1.

5.1.1 Podatki v konfiguraciji

Za implementacijo osnovnih funkcionalnosti mora konfiguracija, ki jo bo generator prejel, vsebovati vsaj podatke o:

- imenu igre,
- simbolih, ki se pojavljajo na kolutih,
- kolutih,
- plačilnih listah ter
- načinu plačevanja plačilnih linij (ali igra plačuje čez vse možne plačilne linije ali za izbrano število plačilnih linij).

Omenjeni podatki so nujno potrebni pri vsaki kolutni igri, da le-ta deluje pravilno. Pri tem potrebujemo ime igre, da lahko tvorimo ustrezna imena razredov in paketov. Ker lahko igra vsebuje več različnih modulov, mora ta konfiguracija vsebovati tudi podatke za ostale module.

Poleg teh podatkov v konfiguraciji obstajajo še drugi podatki, ki niso obvezni pri vseh igrah. Ti podatki so:

- potek plačilnih linij,
- velikost prikazanih dobitkov in
- podatki o dodatnih dobitkih (misije, zbiranje simbolov na kolutih in zbiranje simbolov za dodatno igro).

5.1.2 Delovanje generatorja

Generator programske kode iz dane konfiguracije naredi vse potrebne razrede in jih napolni s potrebnimi podatki. Ti podatki so lahko:

- simboli na kolutih,
- koluti,
- plačilne liste,
- število kolutov,
- velikost koluta,
- način plačevanja linij in
- potek plačilnih linij.

Podatke mora program pravilno pridobiti in vstaviti v programsko kodo za osnovni modul in modul brezplačne igre. Pri generiranju je zaželeno, da bi generirali še:

- modul s pravilnimi podatki o dodatni igri,
- dodatni modul, če je le-ta potreben,
- osnovo za preverjanje delovanja ter
- dopolnitev projekta z Gradle datoteko.

5.2 Razvoj generatorja

Po postavitvi razvojnega okolja smo v le-tem naredili projekt, v katerega smo najprej preko orodja Gradle dodali knjižnico `minimal-json`. Ko smo se prepričali, da knjižnica deluje pravilno, smo začeli z implementacijo generatorja.

Izkazalo se je, da najprej potrebujemo vse podatke o kolutni igri, saj nekatere (npr. imena simbolov) potrebujemo že ob tvorbi začetnih delov kode (na več kot enem mestu). Po pridobitvi podatkov iz vhodne JSON datoteke smo s programom za generiranje naredili javanske datoteke, ki imajo ustrezno strukturo. Vsebujejo podatke o plačilnih linijah, plačilnih listih in kolutih.

5.2.1 Pridobivanje podatkov

Podatke o vseh simbolih najdemo v JSON objektu (del tega je prikazan na sliki 5.2). Tu se nahaja več različnih podatkov o simbolih v igri, saj se le-ti uporabljajo za različne stvari.

Objekt `naturalSymbol` se uporablja za simbole, ki so prikazani na kolutih igre, objekt `substitutionScheme` (prikazan na sliki 5.3) pa nam pove, kateri simboli so lahko na plačilni liniji. Objekt `symbolClassName` (prikazan na sliki 5.4) pove, ali so ti simboli vključeni na plačilni liniji ali ne.

Generator programske kode najprej začne z dešifriranjem objekta `naturalSymbol`. Na sliki 5.5 je prikazana funkcija, s katero dešifriramo simbole iz JSON objekta. Ta objekt vsebuje osnovne podatke o simbolih, ki se nahajajo

```
"symbolSetsList": {
  "symbolSet": [
    {
      "naturalSymbol": [...],
      "symbolClassName": [...],
      "substitutionScheme": [...],
      "name": "primary"
    }
  ]
},
```

Slika 5.2: Različni objekti za simbole

```
{
  "componentSymbol": [
    {
      "symbolname": "WR_1"
    },
    {
      "symbolname": "WR_2"
    }
  ],
  "name": "WW"
},
```

Slika 5.3: Objekt `substitutionScheme`

```
{
  "type": "Include",
  "name": "WN"
},
{
  "type": "Exclude",
  "name": "XWW"
},
```

Slika 5.4: Objekt `symbolClassName`

na kolutih in jih potrebujemo za dešifriranje drugih dveh objektov (`substitutionScheme` in `symbolClassName`). Dešifriran objekt `naturalSymbol` vsebuje podatke o imenu simbola, njegovem številskem indikatorju in vrednosti povečanja dobitka, če zadenemo dobitek s tem simbolom.

```
int i = 0;
for (JsonValue symbol : naturalSymbol) {
    String symbolName = symbol.asObject().get("name").asString();
    symbolSets.put(symbolName, new NaturalSymbol(symbolName, decodeSymbolValue(symbolName), i++));
}
```

Slika 5.5: Funkcije za dešifriranje simbolov

Nato smo dešifrirali `symbolClassName`, pri katerem v objekt shranimo `symbolClassName` ime in tip tega simbola (vključen na kolutih ali simbol ne obstaja na kolutih). V naslednji zanki v ta objekt dodamo še vse možne simbole, ki se pojavljajo na kolutih iz objekta `substitutionScheme` z enakim imenom.

Ko pridobimo objekte s simboli, lahko pričnemo dešifrirati kolute. Podatki o kolutih se nahajajo v objektu `reelGrid`, ki je na sliki 5.6. V tem objektu so lahko podatki za več različnih modulov.

```
"reelGridsList": {
  "reelGrid": [
    {
      "gridMappingList": {...},
      "reel": [...],
      "name": "BaseGameGrid",
      "symbolSet": "primary"
    },
    {"name": "FreeSpinsGrid"...}
  ]
},
```

Slika 5.6: Objekta `reelGridList`

Podatki o simbolu na kolutu so zapisani v objektu `reel`, ki je na sliki 5.7. Pri tem nam vrednost `index` pove, na katerem mestu na kolutu se nahaja simbol, `value` ime simbola ter `weight` število takih simbolov, ki ležijo zaporedoma na kolutu. Podatke o kolutih zapišemo v listo, v kateri hranimo tabelo števil. Funkcija, ki dešifrira kolute, je prikazana na sliki 5.8.

```
"reel": [
  {
    "item": [
      {
        "index": 0,
        "value": "QQ",
        "weight": 1
      },
      {
        "index": 1,
        "value": "JJ",
        "weight": 1
      },
```

Slika 5.7: Zapis dveh simbolov na kolutu

```java
for (JsonValue reelGrid : reelGridArray) {
    //decode reel
    List<Integer[]> reels = new LinkedList<>();
    for (JsonValue reel : reelGrid.asObject().get("reel").asArray()) {
        //decode symbols on reel
        List<Integer> symbolsOnReel = new LinkedList<>();
        for (JsonValue symbol : reel.asObject().get("item").asArray()) {
            int symbolIndex = symbolSetMap.get(symbol.asObject().get("value").asString()).getSymbolIndex();
            int weight = symbol.asObject().get("weight").asInt();
            for (int i = 0; i < weight; i++) {
                symbolsOnReel.add(symbolIndex);
            }
        }
        reels.add(symbolsOnReel.toArray(new Integer[0]));
    }
    reelMap.put(reelGrid.asObject().get("name").asString(), reels);
}
```

Slika 5.8: Koda za dešifriranje kolutov

Za koluti iz JSON objekta pridobimo plačilne liste. Plačilnih list ima igra po navadi več, zato vse možne liste shranimo v mapo, za ključ pa nastavimo njihovo ime. Ta objekt je prikazan na sliki 5.9.

```
"winCombinationSetList": {
  "winCombinationSet": [
    {
      "winCombination": [...],
      "name": "WaysPays",
      "symbolSetUsed": "primary"
    },
    {
      "winCombination": [...],
      "name": "ScatterPays",
      "symbolSetUsed": "primary"
    },
    {
      "winCombination": [...],
      "name": "ScatterPaysFreeSpins",
      "symbolSetUsed": "primary"
    },
    {"name": "ScatterPaysW1"...}
  ]
},
```

Slika 5.9: Izgled `winCombinationSet`

Poznamo več različnih vrst plačilnih list. Vsaka vsebuje posebnosti, zato za vsako naredimo svojo logiko. Logika delitve plačilnih linij je prikazana na sliki 5.10. Med izdelavo smo ugotovili, da imajo nekatere plačilne liste podobno strukturo, zato smo dešifriranje le-teh združili skupaj.

```
switch (winCombinationName) {
    case "LinePays":
    case "WaysPays":
        paylinesMap.put(winCombinationName, decodePaylines(winCombinationSet.asObject(), symbolClassNameMap));
        break;
    case "ScatterPays":
    case "ScatterPaysFreeSpins":
        paylinesMap.put(winCombinationName, decodeScatterPays(winCombinationSet.asObject(), naturalSymbolSetMap));
        break;

}
```

Slika 5.10: Prikaz ločitve plačilnih list

Pri plačilnih linijah imamo v JSON objektu (prikazan na sliki 5.11) zapisano ime plačilne kombinacije, elemente za ponavljanje na plačilni liniji, način plačevanja plačilne linije ter znesek plačila. Odločili smo se, da bomo objekte

s podatki o plačilih kombinacijah enakih simbolov shranili v poseben objekt. Ta objekt je prikazan na sliki 5.12 in vsebuje ime simbola plačilnih kombinacij, minimalno število prisotnih simbolov za izplačilo in največje število le-teh. Poleg tega je v mapi s tem ključem še število pojavitev dotičnega simbola na plačilni liniji ter objekt s celovitimi podatki o plačilu.

```
"winCombination": [
  {
    "symbolWinElement": [...],
    "basepay": 200,
    "name": "W1 W1 W1 W1 W1",
    "payMultiplicationType": "betmult"
  },
```

Slika 5.11: Dobitna kombinacije za 5 simbolov z imenom W1

```
public class SymbolPayline implements ISymbolPayline {
    private final SymbolClassName symbol;
    private Map<Integer, Payline> symbolPaylines;
    private int minimumMatch;
    private int maximumMatch;
```

Slika 5.12: Objekt `SymbolPayline`

Podobna logika velja tudi za dobitke na kolutih, le da se v objekt s podatki o plačilu določene kombinacije na plačilni linij zapiše še število brezplačnih vrtljajev, ki jih dobi igralec.

Sledi obdelava podatkov o poteku plačilnih linij po kolutih. Podatke pridobimo iz objekta `evaluationPatternList`, ki je prikazan na sliki 5.13. V dotičnem objektu so podatki o koordinatah simbola, imenu linije in načinu ocenjevanja linije. Podatke zapišemo v objekt, ki vsebuje ime, način ocenjevanja ter tabelo, kar nam pove, kje na kolutu se mora nahajati simbol.

Nazadnje pridobimo še podatke o velikosti prikazanih kolutov igre. Dobili smo jih tako, da smo iz podatkov o plačilnih linijah poiskali največjo vrednost v tabeli.

```
"evaluationPatternList": {
    "evaluationPattern": [{
        "coord": [{
            "x": 0,
            "y": 1
        }, {
            "x": 1,
            "y": 1
        }, {
            "x": 2,
            "y": 1
        }, {
            "x": 3,
            "y": 1
        }, {
            "x": 4,
            "y": 1
        }],
        "name": "L1",
        "evaluationType": "Payline"
    }, {
```

Slika 5.13: Izgled zapisa plačilne linije

5.2.2 Zapisovanje podatkov

Pri zapisovanju smo najprej naredili predloge datotek, ki jih bomo kasneje napolnili s potrebnimi podatki (slednje smo sicer pridobili že pred tem). Primer ene izmed takšnih datotek je prikazan na sliki 5.14.

```
package #package

#importsUtils

import java.util.LinkedHashMap;
import java.util.Map;

public class #shortNameUtils {

    public static final Map<Integer, SlotSymbolDefinition> SYMBOL_DEFINITIONS = createSymbolDefinitions();
    #createPaylines

    private static Map<Integer, SlotSymbolDefinition> createSymbolDefinitions() {
        #symnolDefinition
    }

    private static #typeOfPaylines createPaylines() {
        #paylines
    }
}
```

Slika 5.14: Predloga datoteke `Utils`

V dotični predlogi se vse vrednosti, ki jih je potrebno zamenjati, začnejo

s simbolom `#`. S tem poskrbimo, da se pri zamenjavi vrednosti v programu ne bi pripetile napake. Del teksta, ki ga bo v datoteki potrebno zamenjati, je potrebno najprej generirati.

Predstavili bomo le primer polnjenja za eno izmed datotek s potrebnimi podatki. Datoteka se imenuje `Utils`. V njej hranimo podatke o simbolih in plačilnih linijah, ki se uporabljajo v več različnih modulih.

Datoteko generiramo tako, da najprej preberemo predlogo v niz, ki ga bomo kasneje obdelovali. V tem nizu nato zamenjamo naslednje podatke:

- `#importUtils` zamenjamo s knjižnicami, ki jih je potrebno uvoziti,
- `#symbolDefinition` zamenjamo s konfiguracijo simbolov, ki se pojavljajo na kolutih,
- `#createPaylines` zamenjamo s plačilnimi listami,
- `#packeg` zamenjamo z imenom paketa in
- `#shortName` zamenjamo s tričrkovnim imenom igre.

Pri tem pazimo, da podatke v datoteko zapisujemo tako, da je končna struktura zapisane datoteke enaka tisti, ki jo že uporabljamo. Slika 5.15 prikazuje končno datoteko `Utils`.

Podobno naredimo tudi za vse ostale datoteke, ki jih je potrebno zapisati: `SlotModule`, `SlotDefinition`, `FreeSpinModule`, `FreeSpinDefinition` in ostale. Na koncu v projekt dodamo še datoteko `Gradle` in `MillionSpinTest`, ki preverja, ali igra igralcu povrne pravilen odstotek denarja, če bi le-ta igro igral neskončnokrat.

5.3 Preverjanje delovanja

Preverjanje delovanja generatorja smo izvajali tako, da smo izbrali kolutno igro, ki ne vsebuje veliko posebnosti. Namenoma smo izbrali igro, ki smo jo

```java
package com.oryxgaming.rgs.gdk.nawa.game.arr;

import com.oryxgaming.rgs.gdk.slot.module.paytable.Paytable;
import com.oryxgaming.rgs.gdk.slot.module.SlotSymbolDefinition;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;

public class ArrUtils {

    public static final Map<Integer, SlotSymbolDefinition> SYMBOL_DEFINITIONS = createSymbolDefinitions();
    public static final KalambaPaytableWaysPay PAYTABLE_SCATTER = createPaylines();

    private static Map<Integer, SlotSymbolDefinition> createSymbolDefinitions() {
        LinkedHashMap<Integer, SlotSymbolDefinition> defs = new LinkedHashMap<>();
        defs.put(0, new SlotSymbolDefinition(0).setSymbolName("S_1"));
        defs.put(1, new SlotSymbolDefinition(1).setSymbolName("S_2"));
        defs.put(2, new SlotSymbolDefinition(2).setSymbolName("AA"));
        defs.put(3, new SlotSymbolDefinition(3).setSymbolName("KK"));
        defs.put(4, new SlotSymbolDefinition(4).setSymbolName("QQ"));
        defs.put(5, new SlotSymbolDefinition(5).setSymbolName("JJ"));
        defs.put(6, new SlotSymbolDefinition(6).setSymbolName("TT"));
        return defs;
    }

    private static KalambaPaytableWaysPay createPaylines() {
        Paytable paytable = new Paytable("Basic");
        paytable.addConfig(new SlotSymbolConfig(0).setMinimumMatch(2).setWinAmounts(15, 50, 100, 200));
        paytable.addConfig(new SlotSymbolConfig(1).setMinimumMatch(2).setWinAmounts(5, 30, 50, 100));
        paytable.addConfig(new SlotSymbolConfig(2).setMinimumMatch(3).setWinAmounts(5, 15, 50));
        paytable.addConfig(new SlotSymbolConfig(3).setMinimumMatch(3).setWinAmounts(5, 15, 50));
        paytable.addConfig(new SlotSymbolConfig(4).setMinimumMatch(3).setWinAmounts(5, 15, 50));
        paytable.addConfig(new SlotSymbolConfig(5).setMinimumMatch(3).setWinAmounts(5, 15, 50));
        paytable.addConfig(new SlotSymbolConfig(6).setMinimumMatch(3).setWinAmounts(5, 15, 50));
        return paytable;
    }
}
```

Slika 5.15: Končna datoteka `Utils` za igro Arms Race

na ročni način naredili že prej, tako da smo poznali vse njene posebnosti. To nam obenem omogoča, da obe rešitvi med seboj primerjamo.

Izbrana igra je Tiger Claws, ki ima le poseben način plačevanja linij (plačuje čez vse možne linije), vendar to posebnost naš generator podpira. Sicer vsebuje tudi zbiranje simbolov v enem vrtljaju, pri katerem bomo morali nekatere podatke vnesti ročno, saj jih z generatorjem ne moremo dobiti.

Za dotično igro smo najprej s spletne strani pridobili JSON datoteko, ki vsebuje vso konfiguracijo igre. Vsebina te datoteke je prikazana na sliki 5.16.

Po pridobitvi JSON datoteke smo v programu nastavili ime igre, JSON datoteko in način plačevanja plačilnih linij. To je prikazano na sliki 5.17.

Potem smo pognali generatorski program. Generiran projekt igre smo

Data	Length	Time
{"body":{"cageCode":"OD1","clientType":3,"languageCode":"ENG","playMode":2,"reconnect":false,"token":"a9a9652c-441b-433e-807a-a7f9849bbf9e","version":"9.2....	218	08:45:42....
{"header":{"mId":1,"cId":1,"name":"Authenticate","code":1,"dType":1},"body":{"currencyCode":"FUN","countryCode":"INTL","userName":"user_a9a9652c-441b-433e-8...	250	08:45:42....
{"body":{"gameCode":"tigerclaws"},"header":{"cId":2,"dType":1,"mId":2,"name":"OpenGame"}}	89	08:45:42....
{"header":{"mId":2,"cId":2,"name":"OpenGame","code":1,"dType":1},"body":{"ursId":1,"gameCode":"tigerclaws"}}	108	08:45:42....
{"header":{"mId":3,"cId":3,"name":"GameEvent","code":1,"dType":1},"body":{"event":"OPEN_GAME","data":{"missionContext":{"missionConfiguration":{"missionsAvail...	56197	08:45:42....
{"body":{},"header":{"cId":3,"dType":2,"mId":3,"name":"GameEvent","code":1,"dId":1}}	84	08:45:42....
{"header":{"mId":4,"cId":5,"name":"Ping","dType":1}}	52	08:46:12....
{"body":{},"header":{"cId":5,"dType":2,"mId":4,"name":"Ping","code":1,"dId":1}}	79	08:46:12....

Slika 5.16: Konfiguracija za igro Tiger Claws [16]

```
String filePath = "E:/Users/SimonTusar/Desktop/tgc.json";
String gameShortName = "tgcMine";
boolean waysPay = true;
```

Slika 5.17: Parametri za zagon programa

premaknili v ustrezen projekt z igrami, saj se igra sklicuje na razrede, ki se nahajajo v tem projektu.

Ker z generatorjem ne moremo določiti simbolov, ki so potrebni za proženje dodatnega dobitka, smo le-te v konfiguraciji na sliki 5.18 nastavili na simbole, ki ga prožijo. To moramo narediti zato, ker konfiguracijska datoteka ne vsebuje teh podatkov, ampak so le-ti del drugega dokumenta.

```
@Override
protected Map<String, AbstractKalambaMissionSymbol> createSymbolMapping() {
    Map<String, AbstractKalambaMissionSymbol> result = new HashMap<>();
    result.put("symbols", new KalambaMissionSymbol("symbols", new int[] { TODO }));
    return result;
}
```

Slika 5.18: Funkcija za nastavitev simbolov, ki prožijo dodatne dobitke

Sledilo je še popravilo vrednosti, ki jo vrne funkcija `getExpectedPayout` na vrednost, ki jo dobimo v Excelovi datoteki (slika 5.19). Tudi tega podatka konfiguracijska datoteka žal ne vsebuje.

```
@Override
protected double getExpectedPayout() {
    return 0;
}
```

Slika 5.19: Funkcija `getExpectedPayout` pred spremembo

Po nekaj začetnih težavah ob zagonu generatorja (nekajkrat smo se zmotili

pri prepisovanju imen funkcij in knjižnic) smo brez večjih težav dobili pravilno generirano kodo igre, ki daje pravilen RTP[2]. To je pokazatelj, da je bila igra ustvarjena pravilno.

Po preverjanju igre Tiger Claws smo preverili še igro z zahtevnejšo konfiguracijo. Spet smo izbrali že znano igro Gates of Babylon. Po generiranju kode smo popravili konfiguracijo za misije (prikazana je na sliki 5.20) in vrednost v funkciji `getExpectedPayout`.

```
@Override
protected Map<String, AbstractKalambaMissionSymbol> createSymbolMapping() {
    Map<String, AbstractKalambaMissionSymbol> result = new HashMap<>();
    result.put("wilds", new KalambaMissionSymbol("wilds", new int[] { TODO }));
    result.put("scatter", new KalambaMissionSymbol("scatter", new int[] { TODO }));
    result.put("medwins", new KalambaMissionSymbol("medwins", new int[] { TODO }));
    return result;
}
```

Slika 5.20: Konfiguracija za misije (pred spremembo)

Ob prvi izvedbi smo ugotovili, da potrebujemo še nastavitev igrane misije, saj vsaka misija vrne drugačen RTP. Preverjanja brez imena misije tudi sicer ne delujejo. Do pravilnega RTP smo prišli šele, ko smo uspešno dodali dodatne funkcionalnosti.

[2]Običajno je RTP kolutne igre okoli 96%

Poglavje 6

Analiza

Z analizo smo skušali ugotoviti, za koliko lahko pospešimo izdelavo igre z avtomatskim generatorjem. Poleg tega smo želeli izpostaviti težave tovrstne analize in našega generatorja.

6.1 Analiza pospešitve izdelave igre

Najprej bomo primerjali čas ročne izdelave že narejene igre s časom, ki smo ga porabili za izdelavo iste igre s pomočjo generatorja kode. Da bi dobili objektivne rezultate, bomo še enkrat ročno naredili isto igro in izmerili še ta čas. S tem želimo izločiti nastale učinke prvega razvoja, ko še ne poznamo vseh podrobnosti igre.

Za ustreznejše preverjanje bomo sprva izbrali preprostejšo igro, ki jo lahko generator pretežno naredi sam, torej brez večjih potrebnih posegov v programsko kodo. Kasneje bomo primerjavo izvedli še za zahtevnejšo igro, ki je program ne more v celoti narediti sam. Z drugim primerom bomo upoštevali manjkajoče funkcionalnosti generatorskega programa, ki jih nismo pokrili s prvim, lažjim, primerom.

Pri izbiri lažje igre imamo na izbiro dve možnosti: Tiger claws in Mammoth chase. Odločili smo se za igro Mammoth chase, saj smo igro Tiger claws že prikazali v prejšnjem delu naloge.

Pri težji igri imamo na izbiro več iger, ki jih najdemo na spletnem viru [17]. Tu smo izbrali igro Gates of Babylon, ki smo jo v preteklosti že izdelali sami, pokriva pa več funkcionalnosti kot ji naš generator trenutno zmore.

6.1.1 Mammoth chase

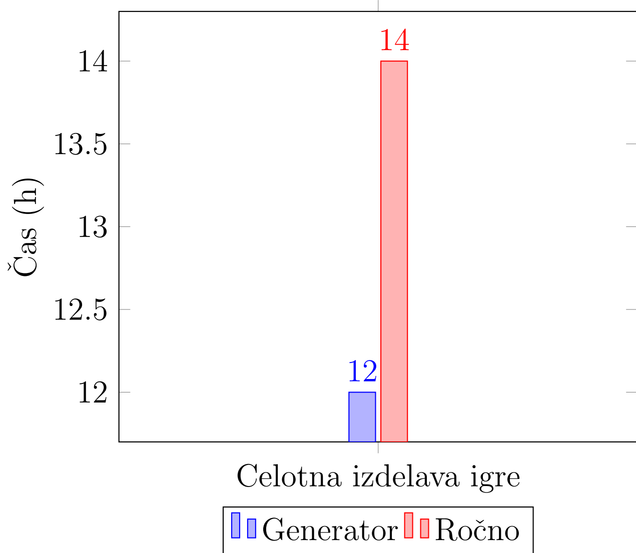
Pri igri Mammoth chase [15] smo za prvo ročno izdelavo porabili približno 60 ur, s pomočjo našega programa pa smo porabili približno 12 ur. Večino časa za izdelavo igre z generatorjem smo porabili za samo preverjanje pravilnosti delovanja same igre.

Ko smo igro ponovno izdelali na ročni način se je porabili veliko manj časa (14 ur), saj:

- ni več potrebe po implementaciji dodatnih funkcionalnosti, ker so te že narejene,
- ni več napak v dokumentaciji, ki jih je bilo potrebno popraviti,
- pravilni podatki za uporabniški so že pripravljeni ter
- ni več potrebe po preverjanju dodatnih funkcionalnosti.

To pomeni, da bi tudi ob prvotnem razvoju z generatorjem porabili precej manj časa kot kaže rezultat. Pri prvi ročni izdelavi je bilo veliko težav in dodatnih funkcionalnosti, ki smo jih morali rešiti in preveriti. Enako pa bi bilo pri avtomatskem generiranju. V primeru druge ročne izdelave te igre se v primerjavi s prvotno ročno izdelavo zato vidi velik časovni prihranek. Generator za ponovno generiranje programske kode zmanjša čas izdelave igre za skoraj 15% v primerjavi z drugo ročno izdelavo. To pomeni, da bi lahko pri izdelavi preprostejših iger prihranili precej časa, kar je razvidno iz grafa 6.1.

V primeru, da bi pa izpustili preverjanje, bi se odstotek pospešitve še povečal. Generator nam omogoča, da celotno igro naredimo v približno 30 minutah. V dotičnem času bi po ročnem principu uspeli le preimenovati datoteke brez priprave potrebnih podatkov kot so plačilne linije, koluti in



Slika 6.1: Celoten čas za izdelavo igre Mammoth chase

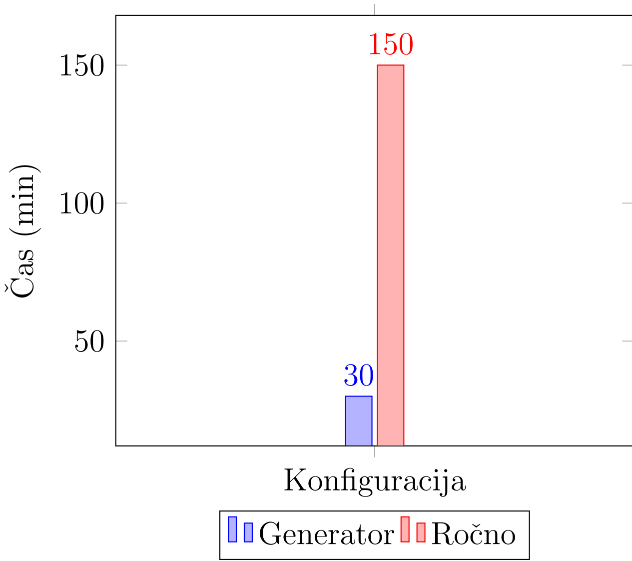
simboli. Razlika je lepo vidna na grafu 6.2. Na slednjem je prikazan samo čas izdelave konfiguracije ob upoštevanju dejstva, da smo v obeh primerih za preverjanje porabili enako časa.

Pri lažjih igrah bi potemtakem z uporabo našega programa za avtomatsko generiranje programske kode prihranili kar nekaj časa, saj bi lahko celotno igro praviloma naredili samo s tem, da bi pravilno nastavili potrebne parametre in pognali program, ki nam generira omenjeno kodo. Poleg tega bi tako zmanjšali določene težave (na primer možnosti za napake), ki se pogosto pojavljajo pri ročnem prepisovanju konfiguracije.

6.1.2 Gates of Babylon

Igra Gates of Babylon [12] je zahtevnejša igra, ki vsebuje kar nekaj posebnosti. Dve izmed teh posebnosti sta:

- V osnovnem modulu igralec na kolutih nabira število pojavitev zamenjalnega simbola. Ko jih zbere dovolj (3), se na kolutu pojavijo le zamenjalni simboli za določeno število iger.



Slika 6.2: Čas izdelave igre brez preverjanja pri igri Mammoth chase

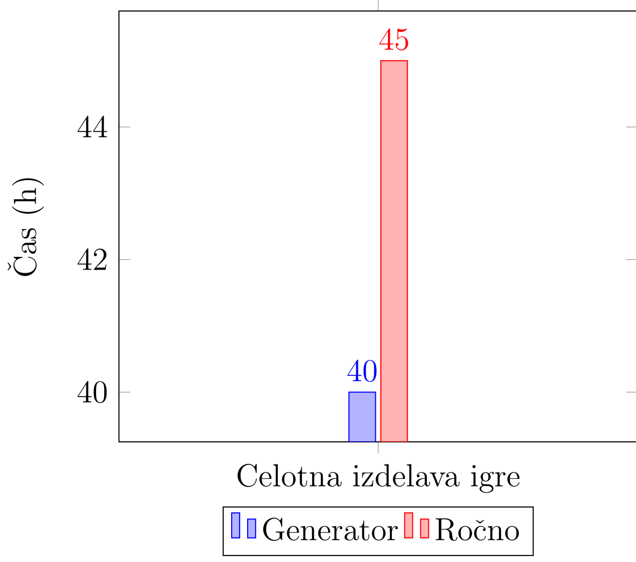
- V igralnem modulu brezplačnih vrtljajev igralec nabira zamenjalne simbole. Ko jih zbere dovolj (3), se celoten kolut do konca igranja tega modula spremeni v zamenjalne simbole.

Poleg teh dodatnih funkcionalnosti ima igra tudi drugačno velikost prikazanih kolutov (5 kolutov, 4 vrstice) ter posledično tudi drugačne plačilne linije. To zahteva dodatno delo. Funkcionalnost različnih velikosti kolutov smo sicer že imeli v celoti implementirano v eni izmed že razvitih iger, zato tu ni bilo potrebno vložiti veliko dela.

Zaradi dodatnih funkcionalnosti pa je bilo potrebno več ročnega posega v programsko kodo, zaradi česar smo za izdelavo igre porabili približno 100 ur. V tem času so zajete tudi težave med implementacijo igre, in sicer:

- napake v dokumentaciji,
- priprava ustreznih podatkov za uporabniški vmesnik,
- igranje igre z uporabniškim vmesnikom ter
- preverjanje dodatnih funkcionalnosti.

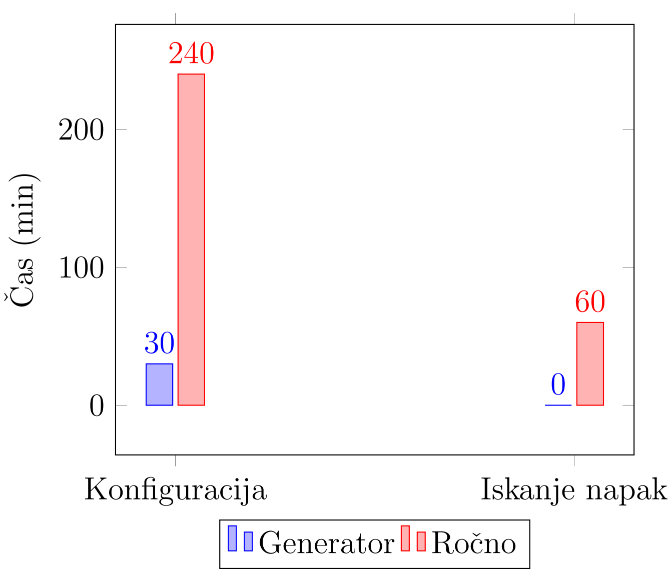
Pri ponovni izdelavi igre smo z generatorjem kode porabili približno 40 ur, predvsem zaradi zahtevnosti dodatnih funkcionalnosti in preverjanja njihovega pravilnega delovanja. Ko smo igro ponovno izdelali ročno, smo porabili okoli 45 ur. Tukaj smo z generatorjem izdelavo pospešili za okoli 12%, kar je razvidno na grafu 6.3.



Slika 6.3: Čas celotne izdelave igre Gates of Babylon

Očitno generator s tvorbo konfiguracije za igro zelo pomaga, saj smo za zahtevnejšo igro porabili praktično enako časa kot za generiranje enostavne igre (Mamoth chase). Pri ročni izdelavi smo porabili veliko več časa zaradi potrebe po konfiguraciji vseh plačilnih linij, ki jih ta igra uporablja. Samo za postavitev projekta brez generatorja smo porabili 4 ure. Za primerjavo: z generatorjem smo porabili 30 minut. Poleg tega smo pri ročni izdelavi naredili napako, zaradi katere smo morali celotno konfiguracijo še enkrat preveriti. To je predstavljalo dodatno zamudo, ki je sicer naključna, a za ročni način pričakovana. Vse je prikazano na grafu 6.4.

Z zapletenostjo igre se tako prednosti generatorja le še povečajo. Seveda se pri zahtevni igri potrdijo tudi vse prednosti, ki smo jih navedli že v pri analizi prejšnje igre.



Slika 6.4: Časa izdelave in iskanja napak pri igri Gates of Babylon

6.1.3 Težave analize

Za tako izvedbo analize žal ne moremo trditi, da nam natančno napove prihranek časa z uporabo generatorja programske kode. Lahko sicer pokažemo, da je generiranje hitrejše kot ročno, vendar tega ne moremo dokazati. Za to bi potrebovali bistveno več primerov, ki bi bili med seboj dovolj različni.

Druga težava je v tem, da smo pri analizi uporabili dve že narejeni igri. S tem smo izločili kakršnekoli napake, ki bi se pojavljale (in se tudi pogosto zares pojavijo) v dokumentaciji. Te bi spremenile čas, ki smo ga porabili pri izdelavi konfiguracije. Poleg tega smo imeli nekatere dodatne funkcionalnosti teh iger že narejene, zato smo jih lahko na podlagi naših prejšnjih izkušenj hitreje izdelali.

6.2 Težave z generatorjem

Prva resna težava z opisanim generatorjem kode kolutnih iger je standardizacija oblike konfiguracijske datoteke. Treba je namreč zagotoviti, da je

dobljena konfiguracija zmeraj natanko taka, kot je predvidena. V nasprotnem primeru generator ne bo znal razbrati podatkov, zaradi česar bo večji del ostal za ročno izdelavo.

Druga težava je nepopolna konfiguracijska datoteka. Lahko se pripeti, da dobimo datoteko za napačno igro ali datoteko, ki še ni dokončna (spremembe simbolov, spremembe kolutov, spremembe plačilnih linij,..). To pomeni, da bi ob koncu implementacije prišli do napačnega rezultata. Napako bi zelo težko odkrili, saj bi se zanašali na pravilne vhodne podatke.

Tretja težava so napake v dokumentaciji. V primeru le-teh bi po dobljeni popravljeni dokumentaciji še enkrat izvedli celoten postopek generacije programske kode, vendar pa bi nam ta še enkrat naredil celoten projekt z igro. Veliko hitreje in enostavneje bi bilo, če bi na izhod izpisali samo spremenjene podatke, ki bi jih nato lahko v programski kodi zamenjali.

Ena izmed možnih težav je tudi, da konfiguracijske datoteke sploh ne dobimo. Ta slabost ni zelo velika, saj lahko igro še zmeraj naredimo na ročni način. Tako generator prinaša le možnost pospešitv, ko tako datoteko pridobimo.

6.3 Ugotovitev

Program za generiranje programske kode zagotovo skrajša čas razvoja kolutnih iger. Ta čas se še posebej zmanjša pri igrah, ki imajo bodisi dolge kolute, veliko število različnih simbolov ali posebno oblikovane plačilne linije. S pomočjo generatorja zelo zmanjšamo tudi možnosti napak pri pisanju konfiguracije, saj je ta proces sedaj avtomatiziran. Poleg teh zelo pomembnih dobrih strani našega generatorja ima ta žal tudi nekaj slabosti. Kot je bilo že omenjeno, so te povezane z ustrezno konfiguracijsko datoteko, torej z zanašanjem na pridobivanje ustrezne datoteke, za katero ni nujno, da je zmeraj na voljo, zanašanjem na popolnost in pravilno strukturo dokumentacije ter z upoštevanjem sprememb dokumentacije.

Poglavje 7

Zaključek

Glavni cilj diplomske naloge je bil optimizirati izdelavo kolutnih iger. To smo uspešno izvedli z izdelavo generatorja, ki je v obliki JSON datoteke zapisano konfiguracijo uspešno pretvoril v programsko kodo za celoten osrednji del igre, pri čemer je bilo treba z ročnim posegom izdelati le določene posebnosti. Pri tem je analiza pokazala, da ključni del izdelave lahko pospešimo.

Pri analizi smo tudi ugotovili, da program, v primeru, da dobimo konfiguracijsko datoteko v pravilni obliki, nima veliko slabosti. Iz tega razloga bi bilo potrebno določiti splošni standard za konfiguracijsko datoteko, saj bi s tem še izboljšali učinkovitost našega generatorja kode.

V generatorju bi lahko sicer izboljšali kar nekaj stvari. Ena izmed bolj opaznih je nadgradnja generiranja programske kode tudi za še druge funkcionalnosti, ki jih igre imajo, a ta trenutek niso podprte. Med take bi lahko šteli na primer pripravo podatkov za dodatno igro. Ker je implementacija tega dela dokaj težavna in vedno vsebuje posebnosti, se tega dela nismo lotili. Dodatne igre delujejo zelo različno, zato je optimizacija izdelave tega modula skoraj nemogoča.

Pomembna izboljšava bi bila, če bi iz datoteke naredili le konfiguracijo in to izpisali. Ta izboljšava bi nam prišla zelo prav, saj v primeru že narejene dodatne funkcionalnosti ne bi bilo potrebno še enkrat narediti celotne igre, ampak bi obstoječo napačno konfiguracijo samo zamenjali s pravilno.

Še ena izmed možnih izboljšav bi bila tudi, da bi nam generator popravil obstoječo datoteko s seznamom vseh iger. Tako bi ob zagonu programa vanjo dodali še igro, ki smo jo ravnokar generirali. To je trenutno še potrebno narediti ročno, vendar tukaj ni potrebno vložiti veliko dela.

Literatura

[1] 10 most popular programming languages in 2019: Learn to code. Dosegljivo: https://fossbytes.com/most-popular-programming-languages/. [Dostopano: 5. 1. 2019].

[2] Apache Maven. Dosegljivo: https://maven.apache.org/. [Dostopano: 20. 2. 2019].

[3] Apache Ant. Dosegljivo: https://ant.apache.org/. [Dostopano: 21. 2. 2019].

[4] Automatic programming. Dosegljivo: https://www.techopedia.com/definition/5638/automatic-programming. [Dostopano: 10. 1. 2019].

[5] Automatic programming. Dosegljivo: https://en.wikipedia.org/wiki/Automatic_programming. [Dostopano: 25. 1. 2019].

[6] J. Bloch. *Effective Java*. Pearson Education, 2017.

[7] Code generation. Dosegljivo: https://tomassetti.me/code-generation/. [Dostopano: 15. 1. 2019].

[8] Code generation & meta-programming. Dosegljivo: https://www.zuehlke.com/blog/en/code-generation-meta-programming-in-embedded-software/. [Dostopano: 10. 1. 2019].

[9] Diagram osnovnih elementov pri programu za izdelavo predlog. Dosegljivo: https://upload.wikimedia.org/wikipedia/commons/c/c7/TempEngGen015.svg. [Dostopano: 2. 2. 2019].

[10] Gradle. Dosegljivo: https://en.wikipedia.org/wiki/Gradle. [Dostopano: 13. 1. 2019].

[11] Igra Dino odyssey. Dosegljivo: https://demo.kalambagames.com/index.html?gameCode=dinoodyssey&playMode=FUN&languageCode=eng&utm_medium=kalamba_website&utm_source=game_list_page&utm_content=PLAY. [Dostopano: 20. 1. 2019].

[12] Igra Gates of Babylon. Dosegljivo: https://demo.kalambagames.com/index.html?gameCode=gatesofbabylon&playMode=FUN&languageCode=eng&utm_medium=kalamba_website&utm_source=game_list_page&utm_content=PLAY. [Dostopano: 20. 1. 2019].

[13] Igra Golden 7 classic. Dosegljivo: https://stagedemo.oryxgaming.com/games/golden-7-classic/HTML5_G7C/fun. [Dostopano: 25. 2. 2019].

[14] Igra Golden dune. Dosegljivo: https://stagedemo.oryxgaming.com/games/golden-dunes/HTML5_GD/fun. [Dostopano: 20. 1. 2019].

[15] Igra Mammoth chase. Dosegljivo: https://demo.kalambagames.com/index.html?gameCode=mammothchase&playMode=FUN&languageCode=eng&utm_medium=kalamba_website&utm_source=game_list_page&utm_content=PLAY. [Dostopano: 20. 1. 2019].

[16] Igra Tiger claws. Dosegljivo:https://demo.kalambagames.com/index.html?gameCode=tigerclaws&playMode=FUN&languageCode=eng&utm_medium=kalamba_website&utm_source=game_list_page&utm_content=PLAY. [Dostopano: 20. 1. 2019].

[17] Igre ponudnika Kalamba. Dosegljivo: https://www.kalambagames.com/games/. [Dostopano: 2. 2. 2019].

[18] Jetbrains. Dosegljivo: https://www.jetbrains.com/idea/. [Dostopano: 12. 1. 2019].

[19] JSON. Dosegljivo: https://json.org. [Dostopano: 15. 1. 2019].

[20] Json example. Dosegljivo: https://json.org/example.html. [Dostopano: 12. 1. 2019].

[21] Kaj je gradle. Dosegljivo: https://docs.gradle.org/current/userguide/what_is_gradle.html. [Dostopano: 14. 1. 2019].

[22] Kalamba games. Dosegljivo: https://kalambagames.com/. [Dostopano: 27. 2. 2019].

[23] Knjižnica minimal json. Dosegljivo: https://github.com/ralfstx/minimal-json. [Dostopano: 12. 12. 2018].

[24] Luknjasti trak. Dosegljivo: https://en.wikipedia.org/wiki/Punched_tape. [Dostopano: 1. 2. 2019].

[25] Luknjasti trak. Dosegljivo: moca.ncl.ac.uk/iomedia/pt1.htm. [Dostopano: 28. 1. 2019].

[26] Najpopularnejše igre v kazinojih. Dosegljivo: https://marketrealist.com/2014/09/must-know-most-popular-casino-games. [Dostopano: 20. 2. 2019].

[27] Notepad++. Dosegljivo: https://notepad-plus-plus.org/. [Dostopano: 20. 2. 2019].

[28] Parsing. Dosegljivo: https://en.wikipedia.org/wiki/Parsing. [Dostopano: 17. 1. 2019].

[29] Pregledovalnik datotek JSON. Dosegljivo: https://sourceforge.net/projects/nppjsonviewer/. [Dostopano: 12. 1. 2019].

[30] Template processor. Dosegljivo: https://en.wikipedia.org/wiki/Template_processor. [Dostopano: 17. 1. 2019].

[31] Tiobe index popularnosti programskih jezikov. Dosegljivo: https://www.tiobe.com/tiobe-index/. [Dostopano: 5. 1. 2019].

[32] Vodnik za kolutne igre. Dosegljivo: https://wizardofodds.com/games/slots. [Dostopano: 20. 2. 2019].

[33] Zgodovina igralnih avtomatov. Dosegljivo: https://www.casinoreviews.co.uk/slots/history. [Dostopano: 20. 2. 2019].