VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

2D STRATEGICKÁ HRA V JAVĚ

2D JAVA STRATEGY GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ NÝVLT
AUTHOR

VEDOUCÍ PRÁCE Ing. MICHAL ZACHARIÁŠ
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá návrhem a implementací strategické video hry v jazyce Java. Součástí práce bude simulace netriviálního počítačového protivníka. V praktické části je popsán postup implementace jednoduché strategické hry War paths.

Abstract

This thesis aims at designing and implemetation of video strategy game. Part of thesis will be aimed at simulation of nontrivial computer enemy. Implementation process of simple computer game War paths will be described in practical part of thesis.

Klíčová slova

Android OS, strategická videohra, umělá inteligence, fuzzy logika, programovací jazyk Java

Keywords

Android OS, strategy videogame, artificial inteligence, fuzzy logic, Java programming language

Citace

Jiří Nývlt: 2D Java Strategy Game, bakalářská práce, Brno, FIT VUT v Brně, 2012

2D Java Strategy Game

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Zachariáše. Uvedl jsem veškeré literární prameny a publikace, ze kterých jsem čerpal.

....
Jiří Nývlt
May 16, 2012

Poděkování

Rád bych poděkoval panu Ing. Michalu Zachariášovy za rady a odbornou pomoc při vývoji aplikace a tvorbě zprávy a kamarádům za pomoc při testování.

Contents

Chapter 1

Introduction

Java language is often blamed for being slow, but nowadays Java proves, that these are mostly just rumors. Java is used for its portability and support for web applications in combination of html or using Java server pages (jsp) technology. It also found use in mobile phone technologies and not only for games. In lots of benchmarks is Java doing better than his main rival C++ language [2], however it's graphics frameworks are not so much effective as current DirectX. Due to it's portability most of video games written in Java are aimed for web pages and mobile phones.

Goal of this thesis is to explore free Java language frameworks with support for 2D graphics and use it to implement simple strategy game. Game will be executable on Unix, Windows and Android operating systems and will have non-trivial computer opponent.

Thesis is divided into 6 chapters. In second chapter I will present design patterns, algorithms and technologies used for implementation of video strategy game War paths. In first part of third chapter I'll introduce features of War paths and its rules, in second part I'll describe and compare currently freely available Java 2D graphics frameworks. Afterwards in chapter four I'll describe core parts of architecture of War paths and process of creating artificial player.

After implementation of War paths I created survey, which is dedicated chapter five. At last in chapter six I will summarize achieved knowledge and discus future of War paths.

Chapter 2

Theory

In following chapter will be introduced evolution of different types of strategy games, techniques and paradigms used in implementing my strategy game War paths. There will be also presented two different approaches of solving simulation of human player in computer games.

2.1 History of strategy games

Video game genres has been evolving since 50', when was developed *Tennis for two* probably the first video game ever. Strategy games appeared around early 80'. First strategy games were mostly only simulation of some board games. Connection of board games and video strategy games proves *Sid Meier: Civilization series*[1], which was based on *Civilization board game*, and *Civilization III: Board game*, which was on the other hand derived from mentioned video game.[9]

Strategy games can be classified as **turn-based** (TBS) or **real-time** (RTS). Turn-based strategies are mostly divided into turns. In each turn player can plan his future moves, which will be executed in given order at the end of a turn. On the other hand in real-time strategies players actions take effect immediately, so it requires better reflexes, but it doesn't need so much of tactic planing.[1]

Goal of most strategy games is to make army, collect resources, build troops from those resources and annihilate your opponent. Ancestor of this military strategy game genre is *Dune II* as a real-time and *Civilization* as turn-based game. But strategy games doesn't necessarily have to be a military games. Goal of construction strategies is not annihilate your opponent, but to manage economy of a complex system. Those games are mostly subclassified as simulation games. For example in 1989 *Simcity* was released by Maxis[2], which is a simulation of creation of a new city. Another popular series of this genre is *Tycoon*. This series started with *Transport tycoon*[3], which was business simulation and nowadays there is more than 30 different types of tycoon games, which are mostly simulations of common problems.

In the present classic real-time military genre is slowly fading. Strategy game genre was mixed with role-playing (RPG) and nowadays most played games belongs to so-called

[1]First *Civilization* was released in 1991 and nowadays *Civilization V* is in stores with release date 2010.

[2]Maxis developed more than 5 versions of *Simcity* and in co-operation with Electronic arts created popular *The Sims series*, famous family simulator

[3]*Transport tycoon*, released in 1994, was designed by Chris Sawyer, who also designed *Rollercoaster tycoon* series, simulation of creation of a new entertainment park

action strategy game genre. Those games are mostly massively multiplayer online (MMO) games, where are two teams fighting each other. Players gains experiences like in RPG and attempts to conquer some strategy point. The first attempt of this genre is custom map for *Warcraft 3* [4] *Defense of the Ancients (DotA)*. In same way as *DotA* was created **Castle fight**, which became paradigm for strategy game designed in this thesis.

2.2 Artificial intelligence

Artificial intelligence (AI) is a simulation of human player to make game playable even when user does not have internet connection to face other human players. Unlike human player AI player has advantage of knowing, what actions human player did. AI must be therefore balanced to be beatable. Strategy game AI algorithms often suffers from it's unconformity to new human player tactics. This kind of issue could be solved by using some learning algorithm, but it requires large effort and computing time, so that's why nowadays in most strategy games is used cheating to get better results.

There are two types of AI, which needs to be handled in strategy games[10]:

- **High-level AI** used to simulate human player. Modern game developing techniques separates this AI into special module, which runs in different thread, then game. Result of this AI are actions, which also use human player like *upgrade attack speed.*
- **Low-level AI** used by game units. Goal of this AI is for example to pick correct path, where shall unit move, or to pick enemy, which will be most vulnerable for units attack.

2.2.1 Fuzzy logic

Fuzzy logic simply describes common problem of a real world. It is also used in mathematics, industry, quantum physics. It is often incorrectly associated with probability or neural networks. **Fuzzy set** instead of allowing only full and no membership like classic set allows also partial membership. To each element of set is assigned **degree of membership** to a set. Degree of membership is determined by **membership function** with range of values in the interval $< 0, 1 >$. This function can be described either by graph or table.

On a figure 2.1 is graph, which describes degree of membership of attributes describing speed of some objects. Objects from this example should be categorized as slow for $speed <$ 40 mph and it membership will gradually reduce to zero as speeds increases to 55 mph. Same approach can be applied on other two attributes. So at 43 mph is object for 0.6 slow and 0.4 medium. Sum of all memberships is 1 as in propositional logic.

[4] *Warcraft 3* is game developed by Blizzard entertainment, which is also creator of other famous titles like *Diablo*, *Starcraft* and *World of warcraft.*

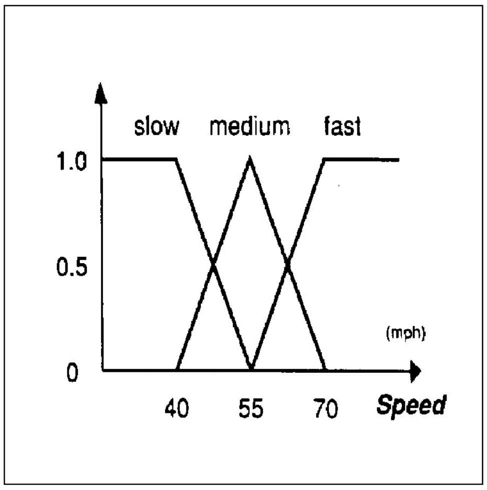


Figure 2.1: membership function

2.2.2 Fuzzy systems

Fuzzy logic principles are used in **fuzzy control systems** (FCS). Besides artificial intelligence FCS can be found in washing machine to control water temperature, digital cameras with automatic light adjustment, robotics, pattern recognition. They are used for it's simpleness of design, implementation and upkeep. FCS consists of parts displayed on figure 2.2

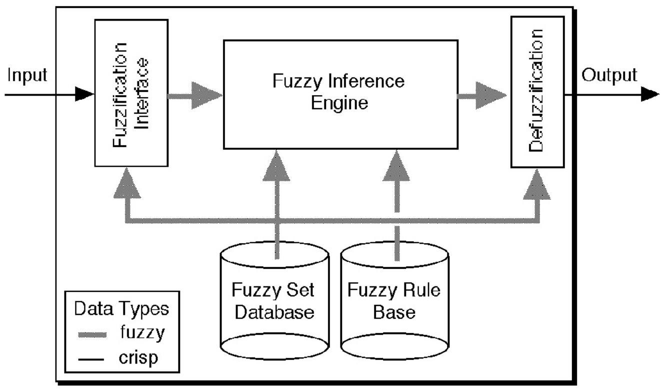


Figure 2.2: Fuzzy system

At first **crisp input**, it can be for example temperature water in washing machine, is sent to unit handling **fuzzyfication**. Fuzzyfication is process during which is the crisp input transformed into degrees of membership in fuzzy sets. Water is in this step assumed as cold. This fuzzy input is then passed to **inference engine**, which uses **fuzzy rules base** to conduct output. Fuzzy rules base is a collection of linguistic terms in form of *IF-THEN* constructions.

Combining rules from fuzzy rules base and fuzzy input from fuzzyfication fuzzy outputs are conducted (water is cold therefore heating element should use lot of power for heating) and pushed into **defuzzyfication** unit, where are all fuzzy sets transformed back to crisp output and spread to affected parts of either machine or application.[5]

Algorithm 1 Rules in fuzzy system

```
if some condition is true then
    do some processing
end if
if water temperature is hot then
    set heating level 2
end if
if water temperature is very cold then
    set heating level 4
end if
```

2.2.3 Minimax

Minimax is an algorithm used to simulate human player in turn-based strategies for two players like chess or tic-tac-toe. Each game situation can be described by acquiring all informations about current game objects. This is called **game state**. Whenever player do some action (move a figure, buy an upgrade), he changes the game state. Applying all actions generates all possible game states to which can opponent get from players current state. Applying this technique recursively gives overview of where is current game heading and which action will be the correct one to perform in current state. Minimax is mostly visualized as a tree (figure 2.3).

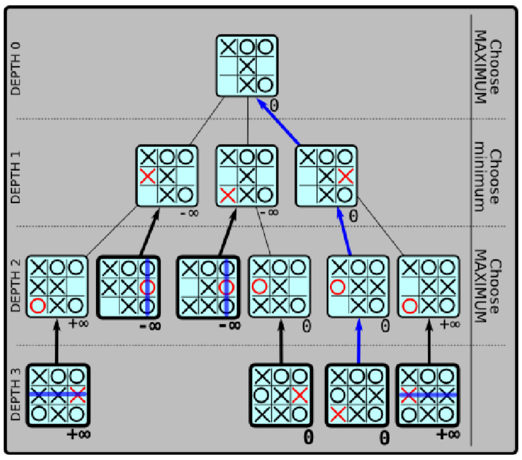


Figure 2.3: Example of minimax tree

Current game state is a root of a tree. By applying all possible actions all game states are generated until game is over, it means no more actions can be applied or player placed three symbols to a row. To leaves nodes is assigned a value sometimes called **score** determined by **heuristic function**. This function describes current player situation. If score is negative it signalize bad game situation for player on turn. Heuristic function is used only on leaves of a tree. Tree is queried from leaves taking maximum value from lower level of tree, if player is on turn, and minimum value, if enemy is on turn. Mostly its impossible to create full game tree until end of a game, so usually is defined a maximum depth to which will algorithm plunge. Algorithm is usually implemented by recursive function described in algorithm 2[6].

Algorithm 2 Minimax pseudocode

```
Node minimax(Node $processedNode$, int $depth$)
if isLeaf($processedNode$) or $depth > MAXDEPTH$ then
  return heuristicFunction($processedNode$)
end if
$depth += 1$
if $processedNode$ isMaxNode then
  $resultNode := -\infty$
end if
for all $child$ of $processedNode$ do
  $tempNode$ :=minimax($child$, $depth$)
  if $tempNode > resultNode$ then
    $resultNode := tempNode$
  end if
end for
return $tempNode$
if $processedNode$ isMinNode then
  $resultNode := \infty$
end if
for all $child$ of $processedNode$ do
  $tempNode$ :=minimax($child$, $depth$)
  if $tempNode < resultNode$ then
    $resultNode := tempNode$
  end if
end for
return $tempNode$
```

2.3 Application architecture patterns

2.3.1 Model-view-controller

This pattern is popular in designing web applications, what proves number of frameworks following this pattern. It separates classes of application into three different modules for data manipulation, application logic and rendering informations to end user. First module **model** takes care about storing data using any available technology like SQL databases or simple XML files. Second module **controller** uses model interface for gathering required informations, performs actions required by end user and passes data to third module **view**. Views work is just rendering informations passed from controller on canvas. It is very necessary for this architecture to keep all those 3 modules separated, otherwise there raises risk of creating dependencies between modules, which will make it hard for upkeep in a future.[4]

2.3.2 Event system

Event system is a pattern for dispatching informations through application. It is composed from `EventListener`, `EventInterface` and `EventData`. `EventListener` can register itself for listening on some event on object implementing `EventInterface`. Registration is done on `EventInterface` by one method `addEventListener(EventListener)`. It adds `EventListener` into `List` container. When required event occurs, `EventInterface` alerts all `EventListeners` based on type of event. Usually there are `EventData` passed to `EventListener` containting more detailed informations about fired event. Pros of this kind of communication is, that object implementing `EventInterface` does not need to know, who is `EventListener` and how he will react on specific event. For example sound module and graphics module can be registered for listening on shoot event of tank. Sound module would react to this event by playing explosion sound, while graphics module plays shooting animation. Tank object does not care, what other modules do with event.[7]

2.3.3 Next event calendar

Next event calendar is data structure used for simulation of some real problem. It is governed by **simulation algorithm**. In simulation is used **simulation time**, which can flow faster than time from real world. This allows to predict events, which will occur in future, in very short period of time.

Calendar is ordered list of **activation records**. Activation record contains following informations:

- activation time in order to know, when event occurs
- priority of event to determine, which event should be executed first, when there are two events with same execution time
- name of event, that is planned to be launched

Any newly inserted activation record must be placed to the right place to calendar according to activation time and priority of event.

In the beginning of simulation process simulation time is initialized, initial events are inserted into calendar and maximum simulation time is set to make simulation deterministic. Simulation algorithm then takes first activation record of calendar. If the activation time of event exceeds the limit, simulation is stopped. Otherwise time is synchronized with activation time of record and required event is executed. Simulation also ends if there aren't any activation records left. During simulation it's possible to collect statistics and use them for any purposes.[3]

Chapter 3

Analysis

In following sections will be introduced basic rules and design of strategy game War paths. In second part of this section will be described and compared free frameworks, that can be used for developing video games using Java language.

3.1 Game design

War paths will be military RTS. Like in classic military RTS goal of this game will be to control objects in two-dimensional space called **scene** and eliminate all objects possessed by other players.

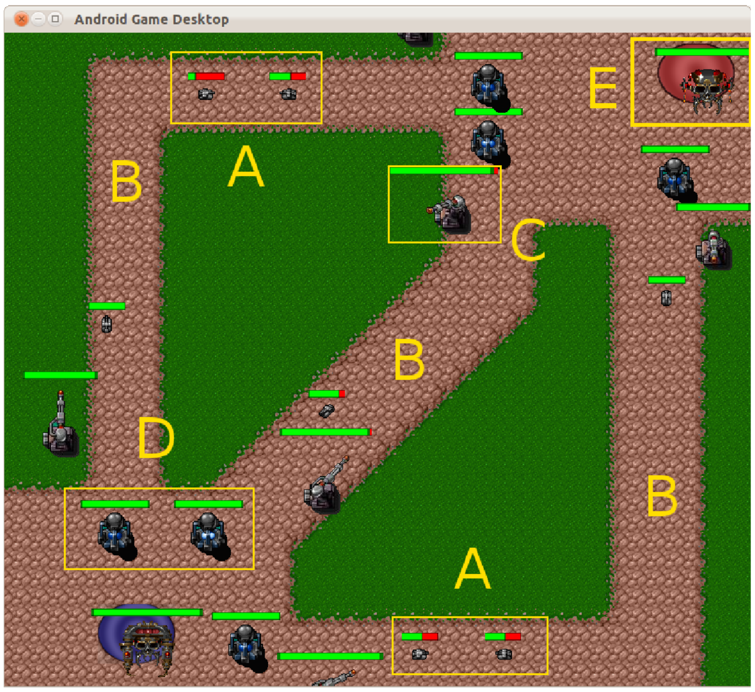


Figure 3.1: First idea about game

First idea in the figure 3.1 shows different types of game objects that can appear on scene. Each game object has **hitpoints** attribute. If this attribute drops bellow zero, object is destroyed and removed from the scene. Each player has one main object called **strategy point** (E), which needs to be protected in order to win the game. Based on settings of level, there will be number of **lanes** (B), which makes connections between two strategy points. In front of strategy points are situated objects called **productors**, which creates new instances of another type of game objects, **attackers** (A).

Role of attacker object is to change its position in scene using lanes towards to opponents strategy point and destroy it. On its way to strategy point attackers meet attackers of other players. They are able to perform actions so-called **attacks** to lower others hitpoints to destroy them. At last there will be objects called **protectors** (C), which can't change position, but they can also attack enemy's attackers.

3.2 Game upgrades

Attackers will have several attributes.

- **Attack speed** (as) attribute will be time between two attacks
- **Movement speed** (ms) will indicate how fast will object move between two points
- Amount of hitpoints (hp) reduced by one attack will be stored in **damage attribute** (dmg)
- Distance from which is object able to attack others is called **range attribute**(rng)
- Time between creation of two instances of attackers is called **spawning time**(st).

Each productor contains information about **enhancement level** (EL) of each attribute. EL is a number determining value of specific attribute for newly created objects. EL value can be increased by action called **upgrade**. Human player can do it by clicking on productor and then choosing from upgrades menu screen. Not all attributes will be connected with attackers. There will be also attribute **income** and attribute **money**. Amount of money attribute will be periodically increased by income value, which'll be also possible to upgrade. To upgrade attribute player must reduce certain value of his money attribute. This value is called **upgrade cost**. Its amount is based on current EL value of upgraded attribute and on specific settings of this attribute. Player wins the game, when his attackers reach last waypoint, which is always position of enemy's strategy point.

3.3 Android 2D frameworks

Game engine represents application layer of common application architecture. Ideal engine for War paths should support these features:

- handling input from user
- organizing game life-cycle
- collision detection support
- easy manipulation with sprites and animations
- tiled map support

- possibility to adapt engine to application architecture, where objects communicates using events
- sound support
- multi-platform support

Currently there are Libgdx, AndEngine, Slick and some other Android 2D free engines. Most common problems of free open source engines is that its developing team does not earn money for its work. Free engines may therefore contain bugs as there is not enough of motivation for developers, but more famous one should be tested by lot of users, whose opinions will be also aspect of picking right engine. Slick engine is very low-level engine with poor documentation and without collision detection support. Use of this framework was rejected it in early begging.

3.3.1 Libgdx

Libgdx is a product of Badlogic Games with copyright owned by Mario Zechner, author of *Beginning Android Games* book. Engine is updated every day and many users joined testing it, which proves users fan page[1], where can be found basic tutorials how to use this engine. Libgdx offers TexturePacker, box2D support, JSON classes loading and many other utilities which will be described in chapter Implementation. It also supports Jogl, LWJGL (OpenGL) and Angle (NVidia 3D Vision) back end libraries for drawing graphics, which provide engine portability on different platforms.

3.3.2 AndEngine

AndEngine offers against Libgdx larger documentation with better tutorials. There is even book[2] about how to create Android game using AndEngine and it's framework with low-level of abstraction. This makes framework easier modifiable, but creates more work to develop some application.

3.3.3 Comparison of frameworks

Libgdx provides easy manipulatable classes with lot of utilities. Even it has almost no documentation, support from it's users is good. It fits all War paths requirements and although it has some chaotic functionality, it's easy-to-use engine. AndEngine provides way larger documentation, but low-level abstraction of interface would cause another unnecessary development. Basically these two engines offers almost same functionality and it's up to users implementation style, which engine will fit for him. In the end was chosen Libgdx as simple way of developing at cost of documentation.

[1]http://code.google.com/p/libgdx-users/

[2]Learning Android Game Programming: A Hands-On Guide to Building Your First Android Game by Richard A. Rogers

Chapter 4

Implementation

Application architecture follows MVC pattern. This pattern is extended by additional part **internal**. Internal part handles loading of all resources and also it provides parsing input from user. Controller module was renamed to logic. Global architecture overview is shown in figure 4.1.

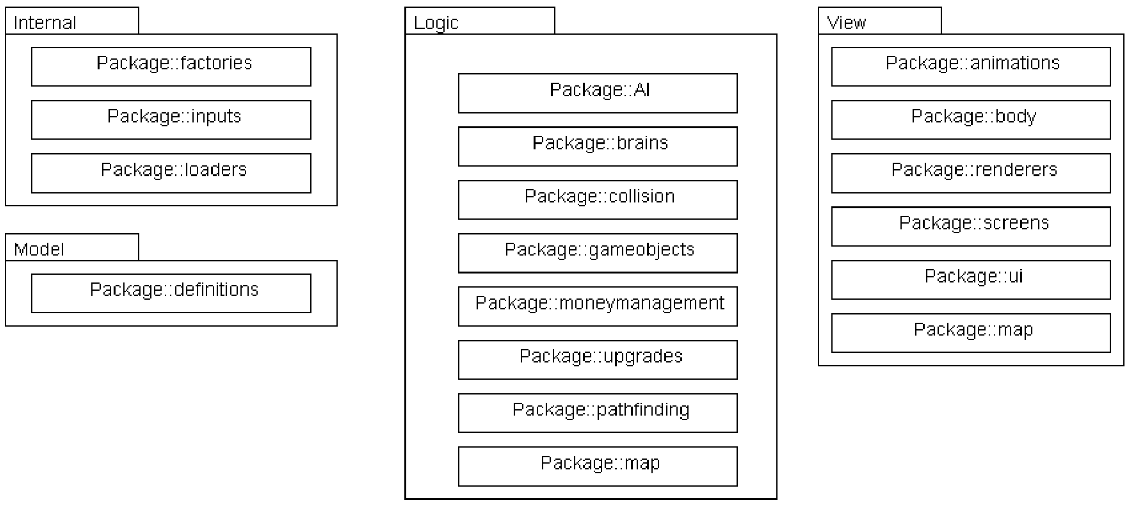


Figure 4.1: Game architecture overview

Game objects are represented by two classes. One is `Body`, which represents view of an object and one is `Brain` for handling objects logic.

4.1 Application life-cycle using Libgdx

Main function of graphics framework is to make abstraction over basic graphics libraries (f.e. OpenGL) and provide interface for loading resources like textures, audio files or other data files. In game development life-cycle is realised by one infinite cycle. In each pass through the cycle is performed update of logic and graphic part of application. Cycle is in case of War paths interrupted when game is over and application is terminated. Usually is application terminated, when user presses the exit button.

Libgdx provides basic interface composed of `backend` class [1] and interface for managing screens. For Windows and Unix based operating systems there is `JoGLApplication` backend class using OpenGL graphic library, for Android version of application there is `AndroidApplication` backend class.

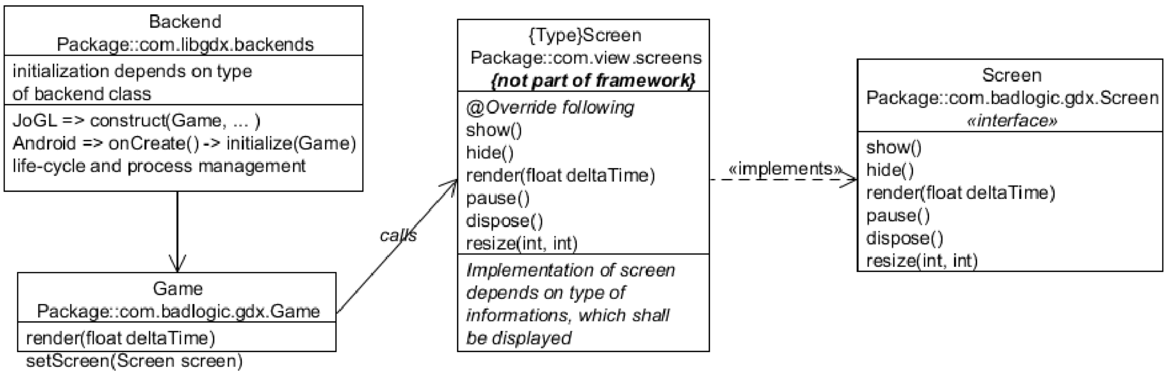

Backend
Package::com.libgdx.backends
initialization depends on type of backend class
JoGL => construct(Game, ...)
Android => onCreate() -> initialize(Game)
life-cycle and process management
Game
Package::com.badlogic.gdx.Game
render(float deltaTime)
setScreen(Screen screen)
calls
{Type}Screen
Package::com.view.screens
{not part of framework}
@Override following
show()
hide()
render(float deltaTime)
pause()
dispose()
resize(int, int)
Implementation of screen depends on type of informations, which shall be displayed
«implements»
Screen
Package::com.badlogic.gdx.Screen
«interface»
show()
hide()
render(float deltaTime)
pause()
dispose()
resize(int, int)

Figure 4.2: Basic Libgdx structure

Connection of core Libgdx classes is shown in figure 4.2. Libgdx backend handles creation of new graphic context to which will be scene rendered and also controls life-cycle. It also raises events that are dispatched using `Game` class to `Screens`. `Screens` are implementations of Libgdx `Screen` interface, that serves events received from `Game` class. Switching of `Screen` classes is done by themselves using `setScreen()` method of `Game` class. Each `Screen` therefore needs to have reference to its parent `Game` class.

4.2 Model

For implementation of model was used `JSONLoader` class. `JSONLoader` can store any instance of class as text into a JSON file format or create new copy of instance from a JSON file in future. JSON (Javascript object notation) is file format easily readable for humans as well as easy for machines to parse it. Using `Reflection` class `JSONLoader` works as a database containing all informations about game settings. Objects attributes were randomly initialized to values shown in table 4.1. Those values must go through balancing process[8], so one unit or upgrade is not too good in comparison with others. It will be done by releasing beta version of a game and using survey.

Table 4.1 Basic units attributes settings

Unit name	Hitpoints	Damage	Spawning time	Attack speed	Range	Ms
Helicopter	400	40	30	3 s	120	80
Tank	600	16	30	2 s	80	80

Units of distance are virtual units recounted by framework to fit each resolution of screen.

[1]Type of backend class depends on operating system on which It's necessary to make deploy of application

4.3 Internal

Internal module consists of loaders, factories and input packages. It handles input from user, provides interface for instancing game objects and takes care about loading all resources before game starts. Loading can't be done in different thread from which applications run, because framework disables process sharing resources.

4.3.1 Input handling

Android devices are mostly equipped with touch screen and number of action buttons. For designing handling of input for user, it is necessary to take in mind restrictions, which its necessary to follow to make game portable for those devices. Moving around map therefore can't be realised like in common computer strategy games by moving cursor to the edge of the screen but by scrolling on map. It is handled by `Cameraman` class. It handles zoom distance of camera according to screen resolution and it disables the user from scrolling out of map.

4.3.2 Factories package

Factories package provides interface for instancing new game objects using `Reflection` class and definitions. There is only one function for instancing new object. It is not necessary to make any modifications to factories, when creating for example new attacker.

4.4 View

Rendering of game scene is separated into two parts. Rendering of game map and rendering of game objects. Libgdx uses special external tool for packing all graphic resources called `TexturePacker`. It clusters more images into one texture, which size is power of 2, and creates text file called `packfile` containing informations about positions, labels and identifier of images. Loading resources is done using `TextureAtlas` class, that represents big image produced by `TexturePacker`. Small images (`TextureRegion` class) can be gathered from atlas according to their labels or identifiers using `findTextureRegion()`.

4.4.1 Game map

For game map rendering was used free external application *TiledMap* editor.[2] Game map is like in chess quadrangle divided into smaller squares of same size called tiles. Tiled map loader accepts images containing small parts of scene called *tilesets*[3] like road curves or cliffs, which can be arranged in graphic interface.

Tiled map editor main function is to create tmx file format[4] containing informations about used tiles and their position in map. To each tileset is assigned range of integer values, that will identify single tiles in each of them. Tmx file and tilesets are loaded by classes provided by framework and rendered in main loop.

[2] available at http://www.mapeditor.org

[3] All graphics used in War paths was extracted from HardVaccum tileset collection available free at http://lunar.lostgarden.com/game_HardVacuum.htm

[4] file format based on xml file format

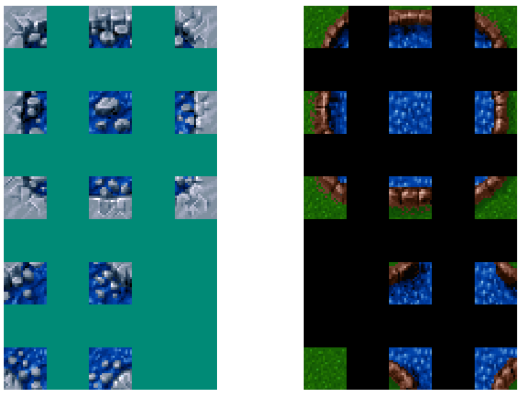


Figure 4.3: Examples of tileset

4.4.2 Game objects, scene and collision

Part of game objects designed for rendering is represented by `ObjectsBody` class. Each `ObjectsBody` reacts on events generated by `Brains` from a logic part and visualise them to end user. Libgdx provides support for moving objects around scene, so it was necessary to extend their classes for War paths's purposes.

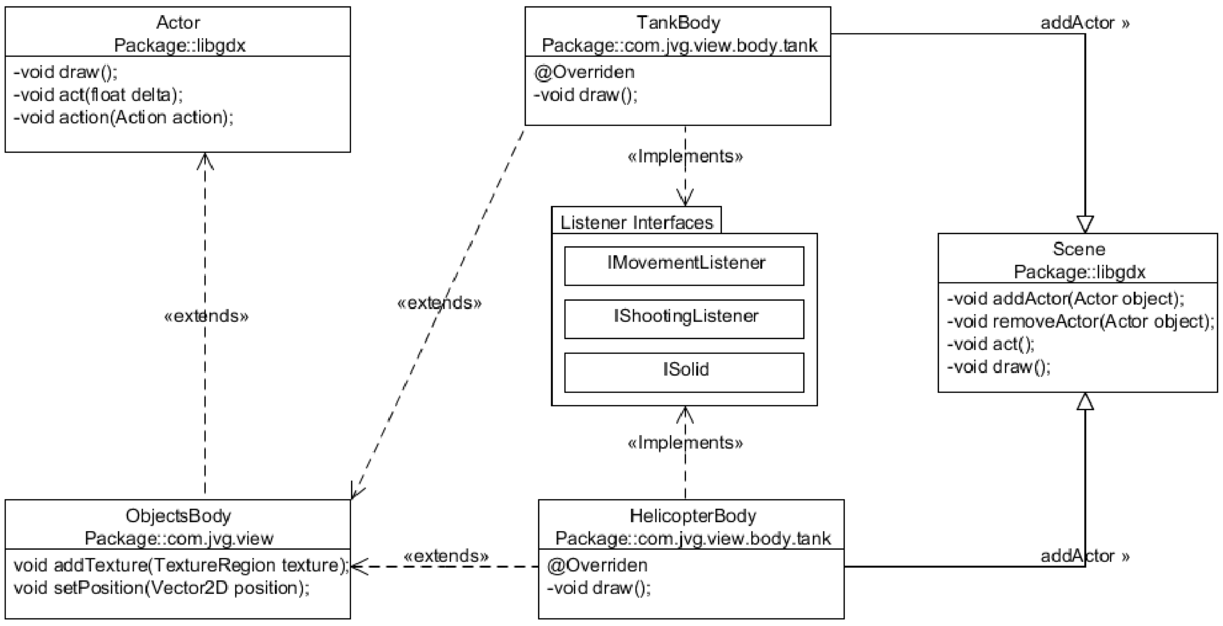


Figure 4.4: Objects body classes structure

In figure 4.4 is shown structure of one `ObjectsBody`. Core of objects body is `Actor` class provided by Libgdx. It has got x and y attribute to keep information about position of body on scene. Its main pros is its `action()` method, which handles moving actor around scene by modifying position attributes. It uses different coordinate system than one used in case of map, unit of this coordinate system is pixel. `ObjectsBody` class inherits from `Actor` class and adds basic functionality like adding textures to draw etc. From `ObjectsBody` inherits specific game object attacker, productor, strategy point or protector. All those objects can choose from various `EventListener` interfaces to be informed about certain events from `Brains`. Each `ObjectsBody` listens at least on one event generated by `Brain` of same type. `ObjectsBodys` are created by factories from internal module and their reference is stored in `Scene` class. Calling `act()` and `draw()` method on `Scene` instance renders all `ObjectBodys`.

4.4.3 User interface

User interface (UI) for War paths is composed of buttons, that needs to be pressed, when user wants to purchase upgrade. Three possible solutions of designing user interface were designed for purposes of War paths. First one is classic strategy game user interface. It divides screen into two parts one for scene and one for user interface.[5]

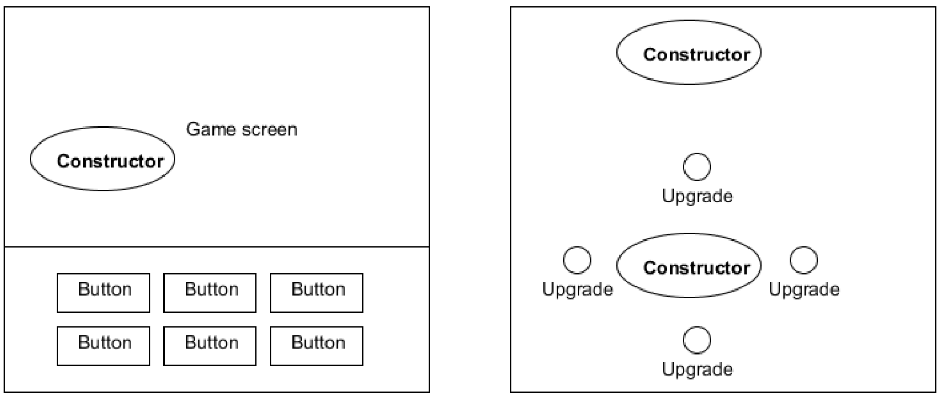


Figure 4.5: Menu designs - Classic and quick buy UI menu designs

Classic design, displayed on left on figure 4.5, is good for screens with big resolutions. All informations to user can be displayed there including informations about money amount and attribute details, but on mobile phones this design leaves almost no space for rendering the scene. Quick buy menu design on the other hand can be used on mobile devices without any issues. Screen is all free for scene, but issue raises, when it's necessary to display other informations than buttons for shopping like price and effect of button. This way of solving UI is anyway used in some mobile applications. Last possibility displayed on figure 4.6 is displaying menu on full screen, which was in the end chosen for War paths.

On game start scene is displayed on full screen. After touching key structure, in case of War paths productor, scene screen swaps with menu screen. Scene can be swapped back by pressing close button. This design is easily controllable but on the other hand removes

[5] This version of UI was also implemented in version 0.09

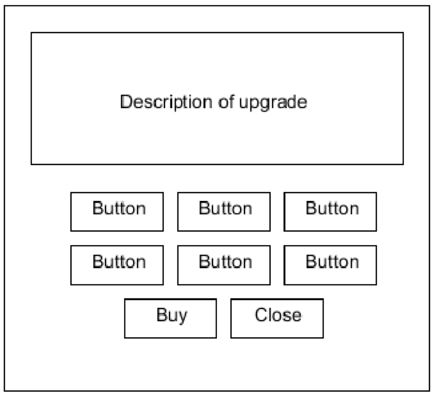


Figure 4.6: Full screen menu

total feeling from a game by hiding simulation, which is necessary to end user and also adds extra actions from user to achieve his goal.

Button is represented by `UIButton` class. Button images must be small in order to place them all on one screen. Clicking them on computer is not big issue, but touching them on mobile phones by finger can be sometimes tricky, therefore places on which `UIButton` reacts as touch are extended over button image to help end user.

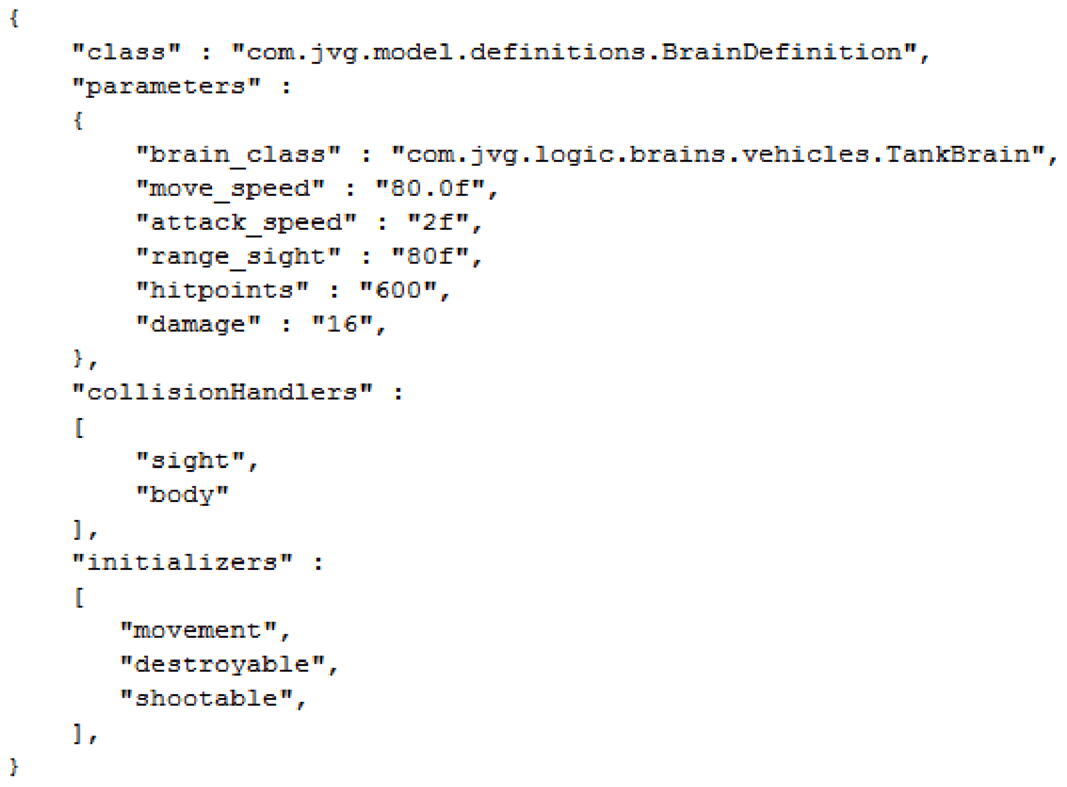
4.5 Logic

Structure of logic representation of game object (`ObjectsBrain`) is similar to structure of its view representation. As it was said `ObjectBody` usually implements `EventListeners`, `ObjectsBrain` serves as source of events handled by `Bodies` and other classes.

4.5.1 Creation of a game object

To make event system works, to `EventInterfaces` must be assigned `EventListeners`. With raising number of game objects putting initialization of event system directly into code would increase code duplicity and it would be also very ineffective. To solve this issue was implemented structure of `Initializer` classes. Each `Initializer` has one main `initialize(Brain,Body)` function, that according to settings in brain definition file shown in figure 4.7, registers listeners on required interfaces.

In the definition file basic attributes definition, collision details and initializer settings are stored. `Initializers` array from previous example indicates, that tank brain will be able to move across scene, will inform other objects, when he is destroyed and will also raise event each time he executes fire action. `TankBrain` and `TankBody` classes must therefore implement interfaces required by initializer. At last it is necessary to specify textures assigned to body in tank body definition file according to implementation of TankBody class.



```
{
    "class" : "com.jvg.model.definitions.BrainDefinition",
    "parameters" :
    {
        "brain_class" : "com.jvg.logic.brains.vehicles.TankBrain",
        "move_speed" : "80.0f",
        "attack_speed" : "2f",
        "range_sight" : "80f",
        "hitpoints" : "600",
        "damage" : "16",
    },
    "collisionHandlers" :
    [
        "sight",
        "body"
    ],
    "initializers" :
    [
       "movement",
       "destroyable",
       "shootable",
    ],
}
```

Figure 4.7: Example of definition of Tank game object brain

4.5.2 AI Implementation

Because the player can't control his attackers implementation of low-level intelligence is fully in control of game logic. Game units follow **way points**, list of map coordinates, which always ends at coordinates of opponents strategy point. Low-level AI is implemented to force attacker to attack enemy game objects in range of sight with the lowest amount of hitpoints attribute. For simulation of human player I've tried two approaches. In first approach I've modified Minimax method to fit real-time application and apply it on War paths. In second I have implemented fuzzy control system for filtering possible actions.

Collision system

Game objects detects each other using Libgdx box2D collision interface. It consists of `World` class representing scene and `Body` classes of various shapes. There are two different shapes, which needs to be placed in collision `World`.

- Body representing physical part of game object. It is a quadrangle usually with same size as game objects texture.
- Body representing game objects range of sight, which is a circle with diameter defined in definition file.

When two bodies collides each other, `World` class generates appropriate events. Coordinates of bodies in world are updated by `ObjectBody` classes.

Fuzzy logic

When human player faces a decision, what upgrade to pick, first thing what he do is, that he checks current situation. If the game is progressing well, he thinks, if is it better to create some reserves in case opponent is planning some action or if it's better to quickly buy some power to try to beat opponent right now. In general human filters possible solutions to get final decision. I decided to try to filter possibilities using fuzzy control systems.

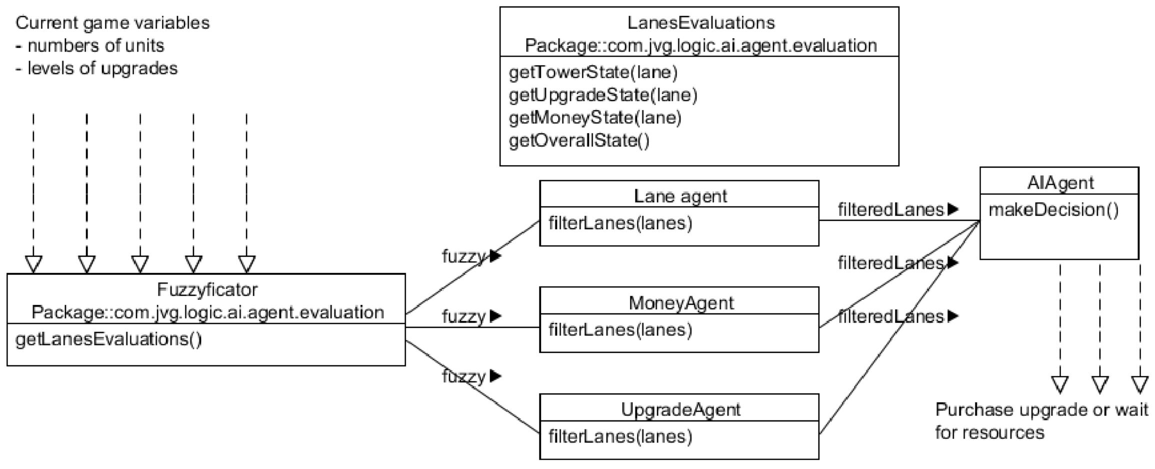


Figure 4.8: Structure of classes included in fuzzy system

In figure 4.8 is shown process of filtering actions. In first step *Fuzzyficator* class transform crisp inputs to fuzzy values. For evaluation of situation on lanes was used version of simulation algorithm next event calendar impoverished of priorities.

Table 4.2 fuzzy values used for evaluation

Fuzzy variable name	Dependency crisp or fuzzy variables	Fuzzy values
defender state	Hitpoints attribute of defenders per lane	0 - weak 1 - average 2 - full
upgrade advantage	simulation evaluation	0 - worst 1 - bad 2 - average 3 - good 4 - best
upgrade cost	price of upgrade	0 - cheap 1 - average 2 - expensive
global situation	player ELs enemy ELs defender state	0 - worst 1 - bad 2 - average 3 - good 4 - best

Evaluation class creates same conditions in simulation engine and counts, how will situation unfold. Second step, fuzzy inference engine, is handled by `AIAgent` class. It has also it's own definition file, in which it's possible to set all fuzzy sets boundaries described in table 4.2. This makes possibility to create different versions of `AIAgent` for example agent who focuses on resources, aggressive or defensive agent etc…

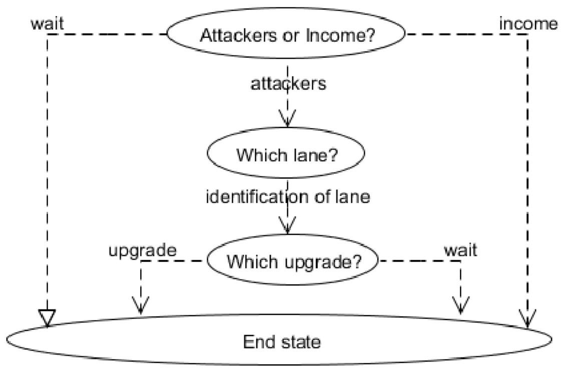


Figure 4.9: Design of decision process of fuzzy system

`AIAgent` filters possibilities using `Sub-agents` classes, which are specialised on a specific kind of issue. This was done for future extension which would allow to assign priority to sub-agents in order to create more types of AI.

First sub-agent is `GlobalAgent`. It checks current game situation without any estimations of future and decides if attacker or income shall be upgraded. In second step `LaneAgent` chooses based on current game situation and states of defenders lane, which should be strengthened in third step by `UpgradeAgent`. All agents computes with fuzzy values like in example in algorithm 3.

Algorithm 3 examples of rules from rules base

```
// example from GlobalAgent
if GlobalSituation in (good, best) AND MoneySituation is bad then
    upgrade income
end if
if GlobalSituation is best AND MoneySituation is average then
    upgrade income
end if
// example from upgrade agent
if GlobalSituation in (good) AND UpgradeState in (good, best) then
    mark as possible upgrade
end if
```

From all possibilities filtered this way is randomly chosen one for next upgrade. In both cases there is a possibility to wait for resources to buy picked upgrade. Last step defuzzyfication is done in `UpgradesManager` class, which stores information about all ELs of all game attributes.

Minimax

I've also tried to simulate human player using Minimax algorithm, last update was in version 0.1.03. As Minimax algorithm is intended for two players turn-based strategy games, its implementation for real-time game War paths must be modified, so game looks like its turn-based. In order to do so, future progress of a game must be simulated. I divided game progress into smaller pieces defined by time interval by default set to 3 seconds.

At the beginning of decision process all current game attributes are duplicated in order to keep game running during simulation. Root of a minimax tree is current game situation and artificial player is on turn. Set of all possible actions player can perform is collected filtering those, which are impossible to perform due to amount of money attribute. At last is added possibility to not perform any action and performing those actions are generated descendants of root node, that represents produced opponents game states. Same step is applied on each opponents game situation.

After opponents step comes the simulation process. Time elapsed from the beginning of game is increased by time interval and money attribute of both players is also increased based on its current EL. Simulation process comes after every opponents action. Those two steps repeats until maximum depth or end of a game is reached. To get evaluation of leaves nodes is used same simulation function as for fuzzy logic. Evaluation is propagated to parent nodes like in classic minimax[6]. War paths then appears as turn-based to minimax even its real-time. Problem I have encountered and a reason, why I didn't use this approach in final version of application, is that its hardly manageable. There is no possibility of creating artificial players with different behaviour and also I did not found a way, how to integrate resources management to it. From a view of performance its bad as well as it requires lot of processor time. On the other hand it proved, that even this method can be used in some kind of real-time games.

[6] Example of tree generated by customized minimax algorithm can be found in appendix A.1

Chapter 5

Survey and game performance

As games are developed for players only possibility to test game performance is let other people play it. I made survey to get feedback about specific areas of game that makes largest impact on quality of gameplay.

- User interface
- Difficulty of artificial player
- Game idea and game itself

I made two versions of game, one for personal computers and one for mobile phones with Android OS and gave the survey to random people (testers) without any knowledge about a game with random PC or mobile devices skills. As there is no tutorial in game it may be hard to find out goal of a game. To help testers orientate in a game I put quick introduction to game into README file attached to version. Whole survey is attached in appendix.

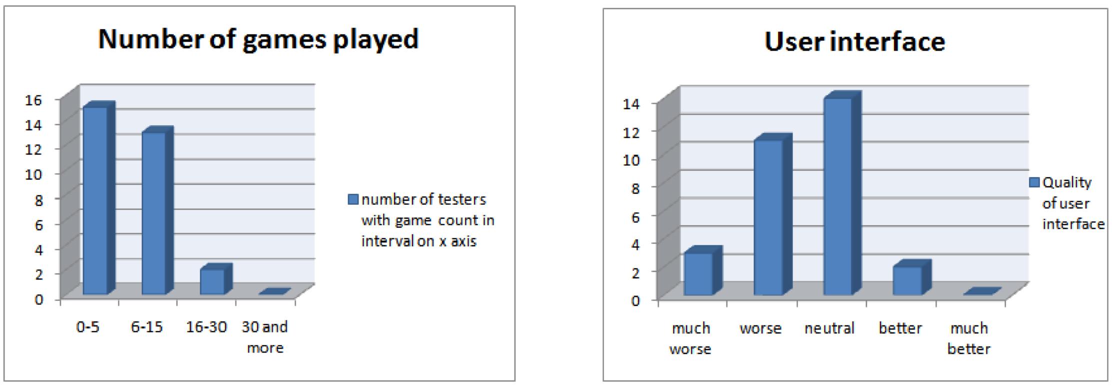


Figure 5.1: Graphs of results from UI part of survey

War paths did not get best rating, which is mostly caused by bad user interface, which proves graphs in figure 5.1. Due to it most of respondents ended they're testing after few attempts. In most cases testers did not understood, what happened after purchasing of any upgrade. There was also problem with back button on Android devices. Testers used it for exiting upgrades menu, which leaded to exiting the game. Some of testers also mentioned

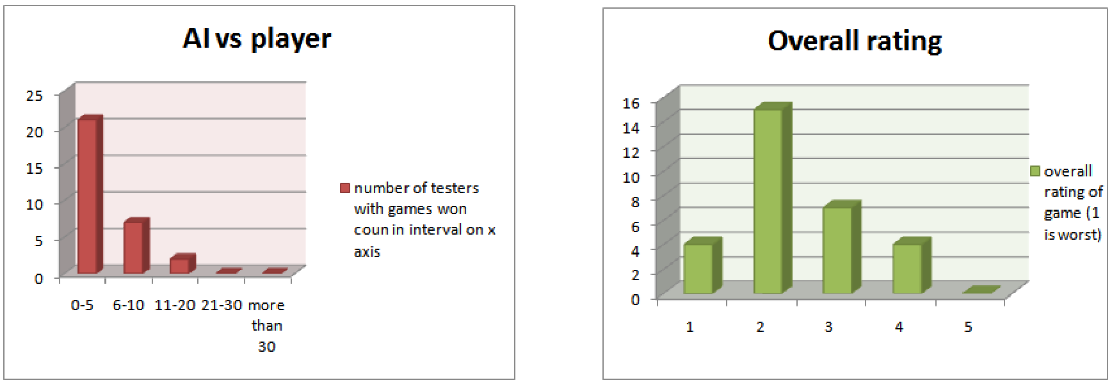


Figure 5.2: O

low amount of units. Large amount of testers was surprised, that strategy point doesn't have any defending abilities. There was also idea to add new repair upgrade, which would set hitpoints attribute of defender back to initial value.

On the other hand even it was alpha testing, no one reported any kind of crash or bug. Only issue was with fonts and Android devices. They were loaded for a very long time and in the end, they were unreadable. Some of devices were for unknown reason probably caused by framework unable to recognize custom fonts imported into framework. I decided to use default framework fonts to avoid those kind of issues. Game was tested on notebook and three android devices. Hardware details and test results are displayed in table 5.1.

Table 5.1 Performance test with device details

Device name	Operating system	Processor	Memory	FPS
Acer aspire 6920G	Windows 7	2 GHz 2 cores	2 GB	60
ZTE Blade	Android 2.3	600 MHz	512	35
ASUS Transformer Prime	Android 4.0	1.5 GHz x 4 cores 700 MHz batery-saver core	1 GB	60
LG Optimus one	Android 2.2	600 MHz	512 MB	38

Column FPS is abbreviation of frames per second and represents count of rendering, which was device able to do in one second. Libgdx life-cycle has limitation to 60 frames per second, so on good machines its impossible to get better performance. For human eye is acceptable game with at least 30 fps in order to see it fluently, so its also acceptable even for mobile phones.

To solve issue with orientation in game it is necessary to extend current UI by adding informations that testers lacked. Most confusing for them was, that they don't see any progress after purchasing upgrade and that there is no information about amount of money attribute required for purchasing upgrade and effect, which will upgrade cause. I designed two possible modification to solve this issue.

First possibility is to use classic UI design from figure 4.5 and combine it with full screen UI (figure 4.6). In classic UI would be displayed current progress of creation of new attackers and possibility to enter full screen UI with upgrades and description. Second one is to put all informations in full screen menu. Choosing between those two possibilities will be part of next release process.

As skills of every players are not on same level in each video game should be possibility to choose difficulty of artificial computer player. In War paths To adjust computer player skills, I must have a look on differences between good and bad player. Bad player unlike good player makes mistakes in decision making and it takes him longer time to make one, because he have no knowledge base about game and neither tactic. Using this pattern on War paths decision making system I can add probability to do a bad decision or longer the period between two actions to decrease difficulty of AI player. Another way of adjusting difficulty is to give to one or more player advantages like more money attribute per interval or to add more percentage bonuses to each upgrade, which will on the other hand increase the difficulty. Graph of survey aimed on artificial intelligence (figure 5.2) showed, that nearly 30% of testers were not able to beat computer opponent. Given the fact that the difficulty of artificial intelligence is also affected by low-quality of user interface, I labelled current implementation of artificial intelligence as advanced and adjusted it to five difficulty levels (very easy, easy, normal, hard and brutal) using mentioned methods.

Chapter 6

Conclusion

Aim of the thesis was to create playable strategy game with artificial player controlled by computer.

In theoretical part I described different approaches, that can be used for simulation of human player, and presented patterns that I used in designing strategy video game War paths. In analysis I tested and compared two different 2D graphics frameworks and decided to choose Libgdx for implementation of War paths. In last part of thesis I described core parts of implementation of strategy game War paths.

I found out that modifying Minimax algorithm is not a good way to simulate artificial computer player in real-time video games. For common decision system Fuzzy proved to be a good solution to create average computer opponent.

Except issues with fonts chosen framework Libgdx proved to be good tool for developing graphics applications for devices with Android operating system mainly thanks to its support of different operating systems. In addition to portability of application, it removes need of uploading and installing translated source codes onto Android device for each test of application, which speeds up the development process.

I made a survey to test quality of a game, which thanks to fact, that user interface doesn't offer enough informations to human player, didn't give a very good results. To solve problems with user interface I designed two possible solutions. Choosing between them will be done in next testing process.

In future I would like to extend current logic part of a game by adding more units and maps for more than two players. Also I would like to add possibility to play War pathsover network and then I would like to offer War paths on Google play for free with adverts.

Bibliography

[1] Apperley, T.: Genre and Game Studies: Toward a Critical Approach to Video Game Genres. *Simulation and Gaming: An International Journal of Theory Practice and Research*, ročník 37, č. 1, 2006: str. 23.

[2] Bourles, J.-Y.; Uso, T.: Java and OpenVMS: Myths and realities. 2007.

[3] Delaney, W.: *Dynamic models and discrete event simulation*. New York: M. Dekker, 1989, ISBN 9780824776541.

[4] Freeman, E. T.; Robson, E.; Bates, B.; aj.: *Head First design patterns*. Sebastopol, CA: O'Reilly, 2004, ISBN 9780596007126.

[5] Ganesh, M.: *Introduction to fuzzy sets and fuzzy logic*. India: Prentice Hall, 2006, ISBN 9788120328617.

[6] Jones, M.: *Artificial intelligence : a systems approach*. Hingham, Mass: Infinity Science Press, 2008, ISBN 9780977858231.

[7] McShaffry, M.: *Game Coding Complete, Third Edition*. Charles River Media, 2009, ISBN 1584506806.
URL `http://www.amazon.com/Game-Coding-Complete-Third-McShaffry/dp/1584506806%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1584506806`

[8] Morris, D.; Rollings, A.: *Game Architecture and Design: A New Edition*. New Riders, první vydání, October 2003, 960 s.

[9] Rollings, A.; Adams, E.: *Andrew Rollings and Ernest Adams on Game Design*. New Riders Games, 2003, ISBN 1592730019.

[10] Walther, A.: *AI for real-time strategy games*. Diplomová práce, IT-University of Copenhagen, 2006.

Appendix A

Minimax

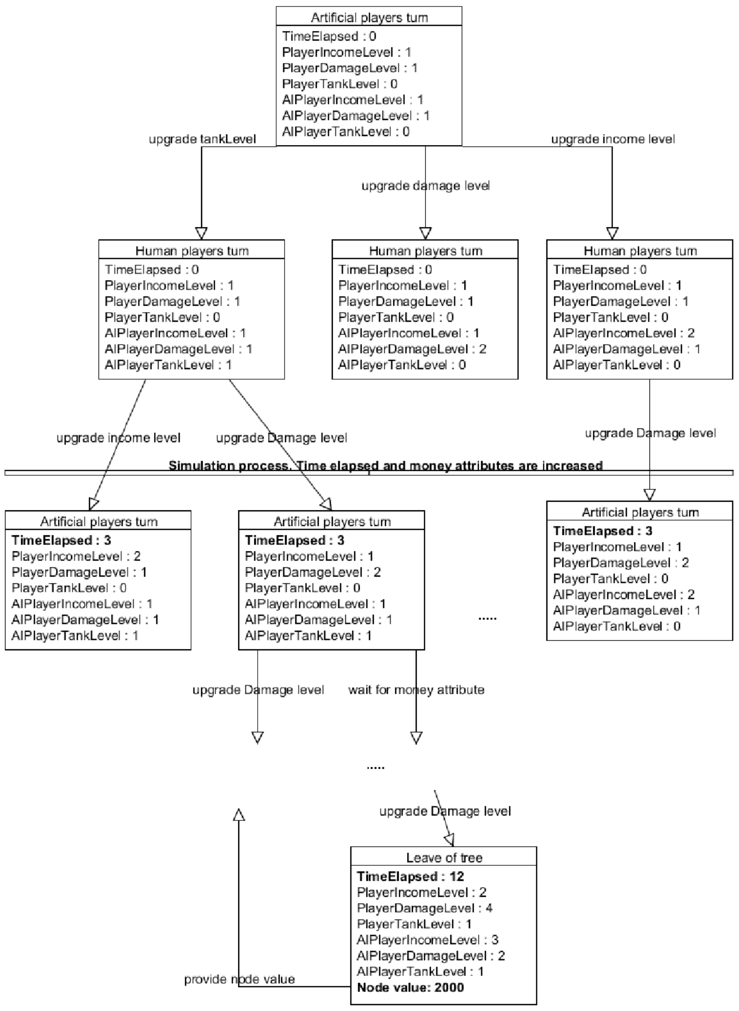
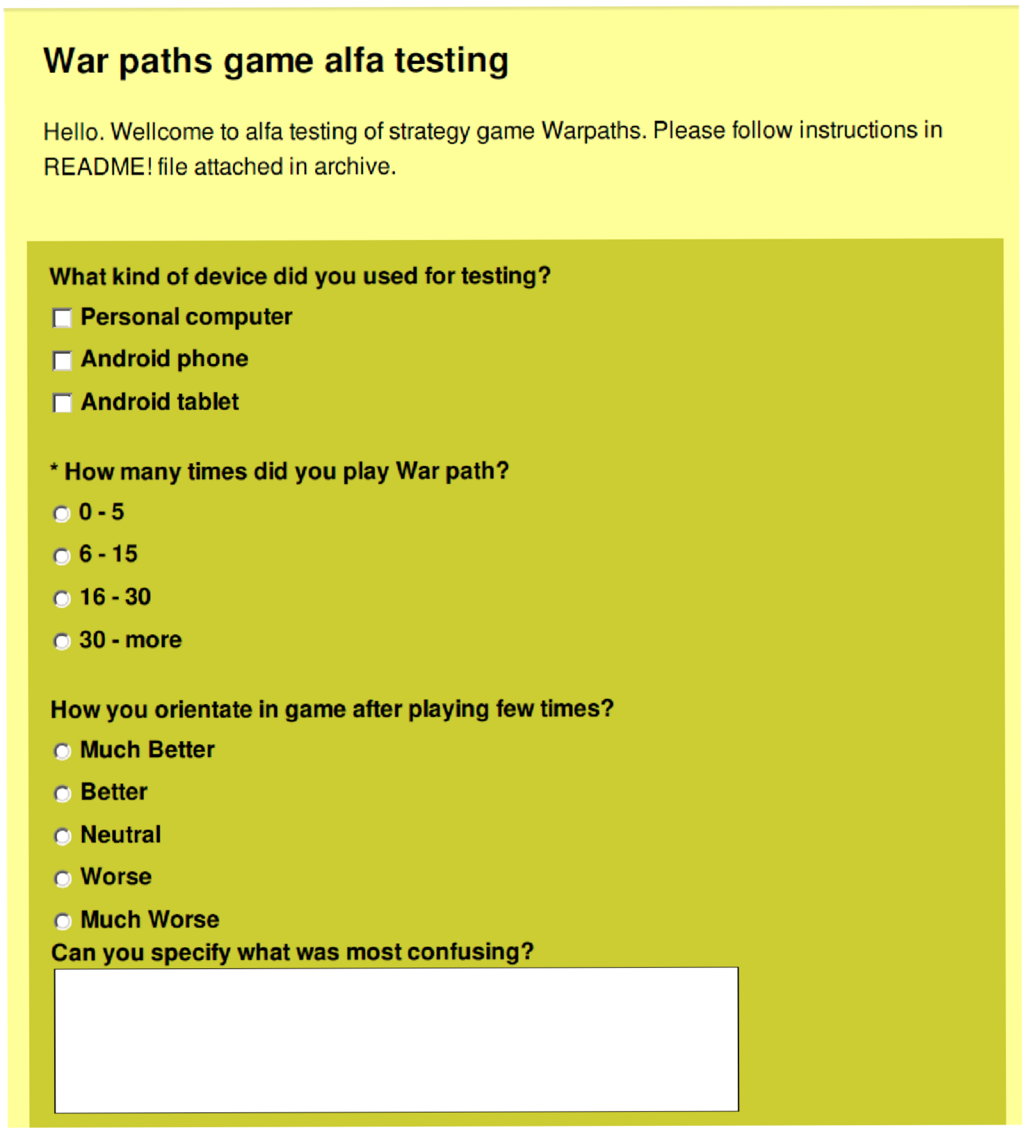


Figure A.1: Tree generated by custom Minimax algorithm

Appendix B

Survey



War paths game alfa testing

Hello. Wellcome to alfa testing of strategy game Warpaths. Please follow instructions in README! file attached in archive.

What kind of device did you used for testing?

- ☐ Personal computer
- ☐ Android phone
- ☐ Android tablet

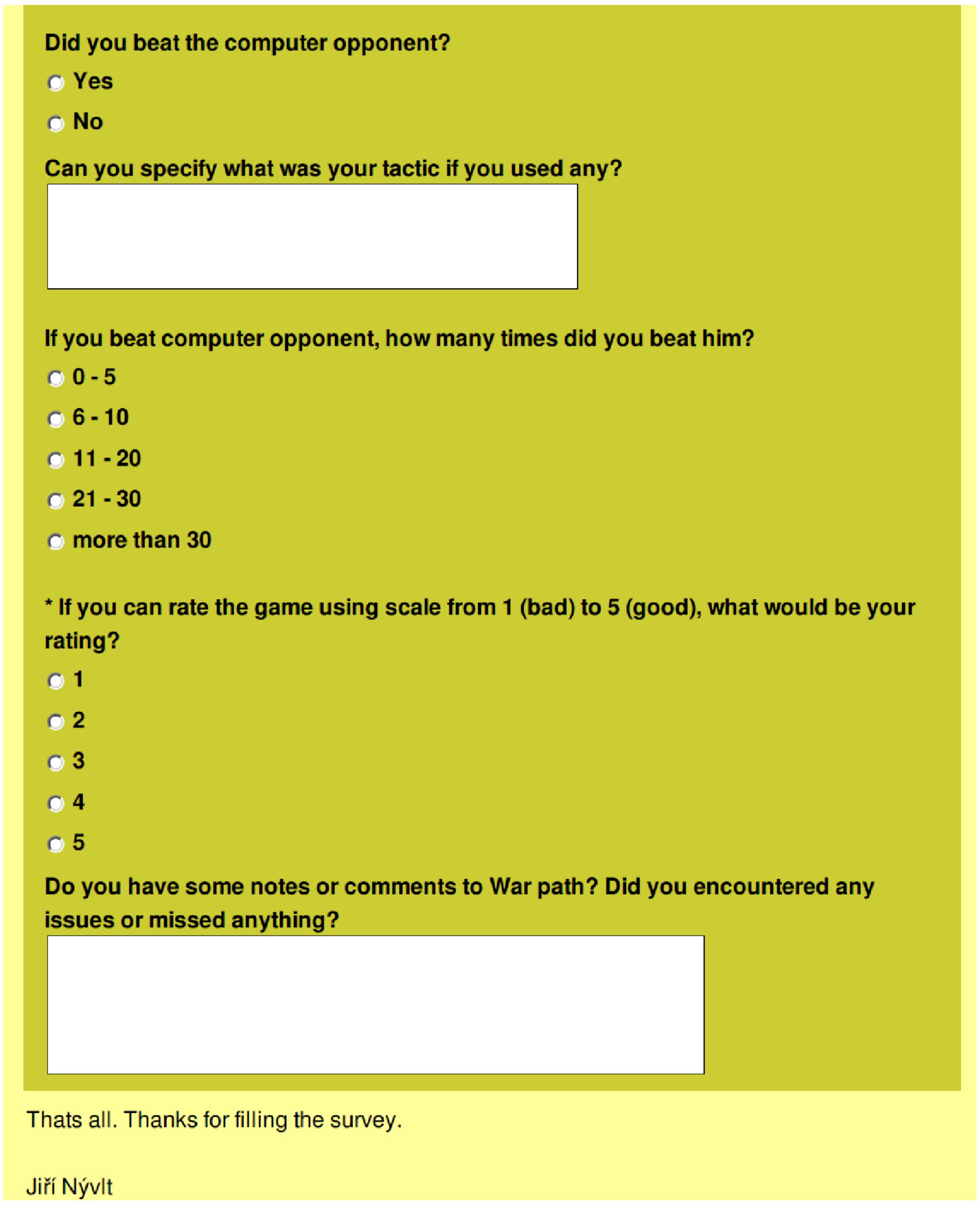
*** How many times did you play War path?**

- ○ 0 - 5
- ○ 6 - 15
- ○ 16 - 30
- ○ 30 - more

How you orientate in game after playing few times?

- ○ Much Better
- ○ Better
- ○ Neutral
- ○ Worse
- ○ Much Worse

Can you specify what was most confusing?



Did you beat the computer opponent?

○ Yes

○ No

Can you specify what was your tactic if you used any?

If you beat computer opponent, how many times did you beat him?

○ 0 - 5

○ 6 - 10

○ 11 - 20

○ 21 - 30

○ more than 30

* If you can rate the game using scale from 1 (bad) to 5 (good), what would be your rating?

○ 1

○ 2

○ 3

○ 4

○ 5

Do you have some notes or comments to War path? Did you encountered any issues or missed anything?

Thats all. Thanks for filling the survey.

Jiří Nývlt

Appendix C

Content of CD

Structure of directories is following:

- **/thesis/src** directory contains source files for thesis
- **/thesis/pdf** directory contains `.pdf` version of thesis
- **/warpath/android** directory contains `.apk` file for Android OS devices
- **/warpath/portable** directory contains required `.jar` file with translated sources and data directory with data resources
- **/warpath/versions** directory contains archives with different versions of the game as it was implemented.
- **/warpath/sources** directory contains source two Eclipse projects. One for desktop version of a game and one for Android OS. Sources are linked so changes are done only in desktop project.

C.1 How to run the game

C.1.1 Android OS

To install game on Android OS device follow next steps:

1. On Android device go to Settings-¿Applications menu and allow Unknown sources.
2. Connect device to PC using USB cable.
3. On Android device click on Turn on USB storage.
4. Copy `warpath.apk` file from `/warpath/android` directory anywhere onto device SD card.
5. Disconnect device from PC.
6. Find `warpath.apk` file on your SD card and choose install option.
7. Game icon should appear in menu.

C.1.2 Windows OS

To run game on Windows it is necessary to have Java runtime environment (JRE) installed. It can be downloaded for free from `http://java.com/en/download/index.jsp`.

1. Copy content of `/warpath/portable` directory anywhere on computer.
2. Double click War paths.jar file.
3. Other possibility is from command line. Go to command line and go to game root directory.
4. Execute following command: `java -jar warpath.jar`

C.1.3 Unix based OS

For unix based systems is also necessary java runtime environment or open java. To install it execute following command: `sudo apt-get install openjdk-7-jre`

1. Open command line and go to game root directory.
2. Execute following command: `java -jar warpath.jar`