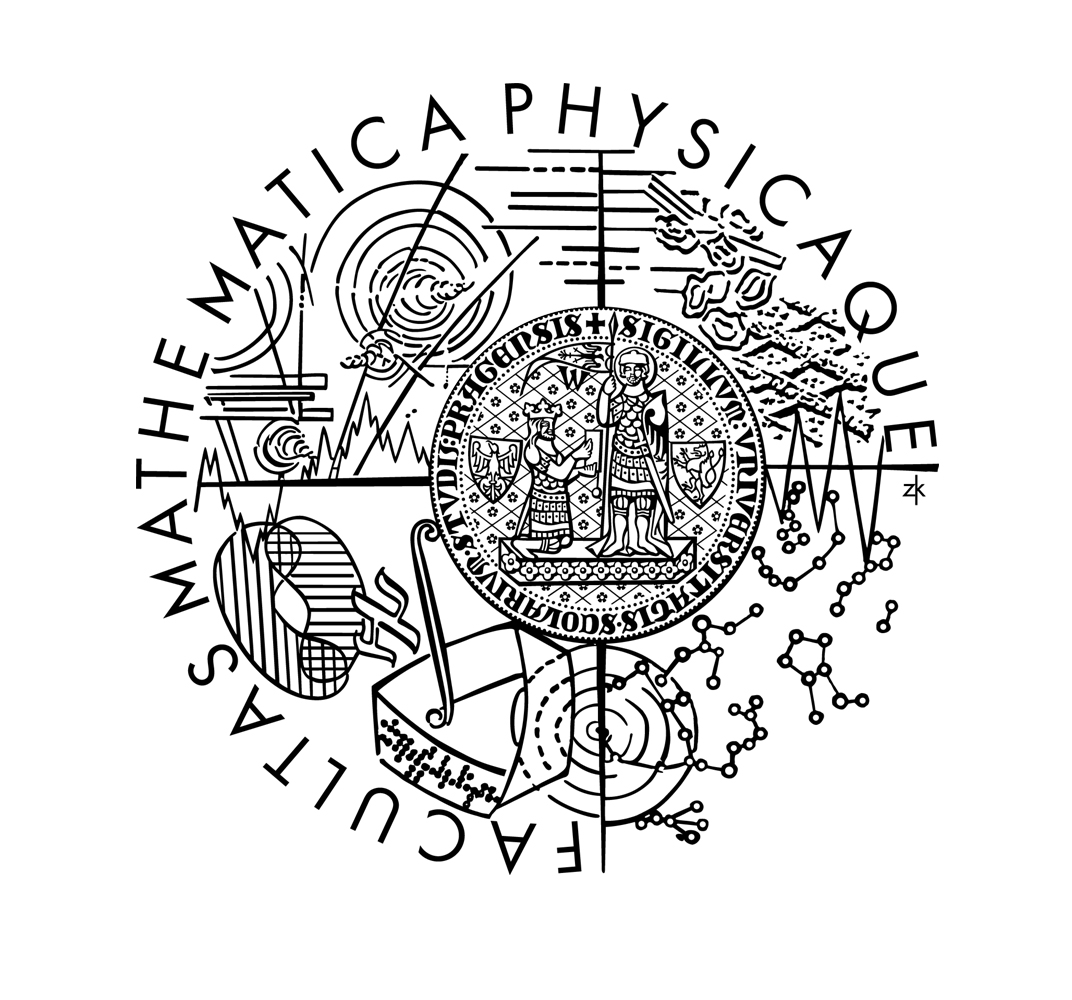
Charles University in Prague

Faculty of Mathematics and Physics

**BACHELOR THESIS**



Štěpán Havránek

**3D action game in a bizzare city**

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: Mgr. Tomáš Balyo

Study programme: Computer Science (B1801)

Specialization: Programování Bc. R4 (NIPR4B)

Prague 2013

[Sample: Bound Sheet – a signed copy of the "bachelor thesis assignment". **This assignment is NOT a part of the electronic version of the thesis. DO NOT SCAN**.]

[Sample: Here you may thank whoever you wish (the supervisor of the thesis, the consultant, the person who lent the software, literature, etc.)]

[Sample: A page with a solemn declaration to the bachelor thesis]

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date............ signature

Název práce: 3D akční hra v podivném městě

Autor: Štěpán Havránek

Katedra / Ústav: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Tomáš Balyo, Katedra teoretické informatiky a matematické logiky

Abstrakt: [abstract of 80-200 words in Czech, but not a copy of the assignment of the bachelor thesis]

Klíčová slova: [3-5 keywords in Czech]

Title: 3D action game in a bizzare city

Author: Štěpán Havránek

Department / Institute: Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: Mgr. Tomáš Balyo, Department of Theoretical Computer Science and Mathematical Logic

Abstract: [abstract of 80-200 words in English, but not a copy of the assignment of the bachelor thesis]

Keywords: [3-5 keywords in English]

Contents

[Introduction 1](#_Toc355108233)

[1. The game 1](#_Toc355108234)

[1.1. Detail description and rules 1](#_Toc355108235)

[1.2. Similar games 3](#_Toc355108236)

[2. Implementation 3](#_Toc355108237)

[2.1. Language and libraries choice 3](#_Toc355108238)

[2.2. Program architecture for real-time game 3](#_Toc355108239)

[2.3. Space and the game world 4](#_Toc355108240)

[2.4. Boxes, tools and action objects 6](#_Toc355108241)

[2.5. People, reflexes and tasks 6](#_Toc355108242)

[2.6. Opponent, task planning 6](#_Toc355108243)

[2.7. Player and game controls 6](#_Toc355108244)

[2.8. Settings and menus 6](#_Toc355108245)

[3. User documentation 6](#_Toc355108246)

[Bibliography 6](#_Toc355108247)

# Introduction

There are many of action games with 3D graphical visualisation made. Main reason for start using synthetized 3-dimensional space was to bring more realistic feeling from the game to the player. Nowadays developers and designers are trying to make better and better simulation of this world using 3D. Aim of this paper is different from these ideas. It deals with game situated in space which doesn’t follow basic physical laws of our world.

Let’s figure out a game that maybe looks like a classical 3D. But parts of game map are connected to each other as a generic graph. In this game you can go straight until you reach your first position. But you don’t come from the back of your original stand at all. For example you can come from the right or from any other direction. And this is the world of our game.

Player’s goal will be to occupy the entire city. He must go to all of its quarters and capture it one by one. His enemy has exactly the same objective. Because of that, both of them have leave captured quarters guarded by their friends. The one, who first orients in the map and gets all parts of map in his property, wins

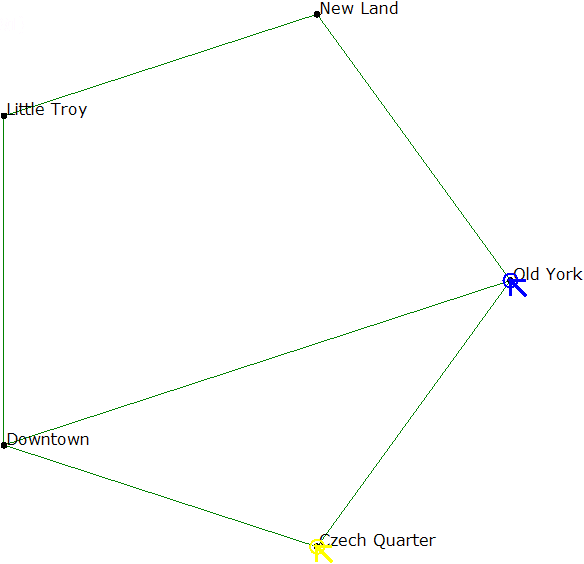
This paper mainly describes implementation of whole action game situated in this introduced space. It begins with model of the program based on Microsoft XNA Game studio [1]. We will go thru real-time programming issues, data representation, used algorithms or modified versions of well-known algorithms for out specific case. And end station of programming part will be implementation of AI for player’s opponent. During this part we will point out several scopes for further development too.

In second part will not be omitted user documentation for creation we make. Reader will find out how to set up and, of course, play the game.

# The game

## Detail description and rules

The city we are playing in is divided into separate quarters. Each of them has its unique name (ex. Downtown). Some of these quarters are connected to some of the others. Together they form a graph. The town graph is always continuous. But degree of one quarter can be only 1.



Picture 1 Town quarters make a graph

Every quarter has somewhere inside flag or empty flagpole. This indicates who the owner of this quarter is. Your flag means that this quarter is in your property. Otherwise the quarter can belong to your opponent or to nobody. Either way is good advice to capture it. The goal of the game is to have captured all quarters in town. When you have or your opponent has reached this objective the game is over. The game begins with one player’s quarter and one opponent’s. Rest of town is without an owner.

In occupied quarters are gradually appearing new guards. They have only one thing to do. When the enemy comes into this quarter, he becomes the target of the guards. Number of guards per one quarter is limited and if you capture opponent’s quarter, his guards will stay until you or your guards kill them. Problem is that limit for guard in one quarter is for sum of all guard – yours and opponent’s. So if you capture quarter full of enemy guards, yours wouldn’t appear until you kill at least one.

How to kill somebody? You can always use your hands, but it’s not recommended approach. It’s not practical. There are four categories of guns:

1. For everybody
2. For guards
3. For players
4. Only boxed ones.

Guns from first category have everyone at start of the game. These from fourth category are available only in boxes. Attention, not only you can take guns from boxes. Your opponent will do it too.

Since we’ve got guns and our enemies have guns it’s necessary to use them. Except guards you will need to shot right the opponent several times. Because when the opponent gets killed he loses all of his quarter except one if he has at least one. And in the one he will appear again alive. If he doesn’t have any quarter, the will show in some empty one. The same thing will happen to you if you get shot. Again, if you lose your quarters by getting shot, your guards will stay there. Only do not appear new ones.

## Similar games

Portal

Z

San Andreas

# Implementation

## Language and libraries choice

## Program architecture for real-time game

Programming real-time applications is other discipline than the other types of software. High emphasis is placed on early response to user input and apparent continuity of episodic process. In other words the game must be able to react and compute its routine at least twenty five times per second.

Since the process has to be fast we need to do some calculations only approximately or asynchronously. Both of these techniques we will use in out game.

Now let’s see, how to make a game architecture for our software. We adopt practices from XNA. It provides prepared process model for whole game. First we need initialize our components, second load all needed content because loading can be very slow operation. Then comes the main game loop between updating the game logic – moves characters, performs actions, etc. and drawing the scene. And at the end, as soon as the game logic decides the game is over, we end the main loop, unload loaded content and do whatever we want. For example exit application or restart.



Picture 2 Game life cycle diagram

It is good idea to have this process distirbuted into separate components. Because of clarity. We have several smaller modules running according the diagram metioned above: Town, Player, Opponent. Moreover the town component ditributes these operations into quarters. And these into walkers, flying bullets, etc.

## Space and the game world

Before we begin model the bizzare world as it was defined, we can prepare some basic building elements. We assemble the world hierarchically and up to specific level we can ignore that the result won’t be placeable into standard vector space. For the second debasement of the problem we consider only two dimensions. The third, height, will be added later only in selected functionalities. We don’t need it everywhere. Finally two-dimensional processing will be faster and that’s what we need.

At the bottom of the space hierarchy we define geometrical elements: line segment, triangle and quadrangle (convex). Everything in our space will be based on quadrangle. Or more precisely every object in the game has projection into two-dimensional space as convex quadrangle. These quadrangles are used for collision detection.

Quite often we need check if two objects are in collision. For example if the bullet hits the man. So we take their projection into quadrangles and compute the collision. Our way to do that is split the quadrangles into two triangles and check them for collisions – four subprocesses. And last thing to catch collision is to compute if two triangles collide. This is simple: we split the triangles into three line segments and find out if any of them is crossing any from the second triangle – nine subprocesses. And we must not forget that for us one triangle inside the second is collision to.

Considering game logic the use of quadrangles isn’t best way to represent base of objects in the game. Quadrangle is defined by four points and it can be little bit confusing if we imagine that we have prepared 3D model (ex. robot) and want to insert it into the game. Should we define all four corner point and scale model to fit corners into created quadrangle points? No. Better add next level in space hierarchy. We will define game object. Game object is structure ready to use for 3D models and it’s simply enough to be still in two-dimensional space. For work with varied 3D models we will use their block shaped bounding box with edges parallel to axes of three-dimensional space. Bottom base of this cuboid is rectangle. And this rectangle is represented by game object. Game object carries information about position, size and azimuth (rotation). The right question here is: what is the position? Is it information about xy-coordinates in simplified two-dimensional space? Or are we now in our bizzare world and position is some kind of description of location in there. The second option is right. Game object, as the name says, describes base of every object in the game. So it has to carry full information about location in our result space.

Now it’s time to say how to represents our bizzare space. After all what are our technical possibilities? We can display on the screen set of objects variously transformed by position in three-dimensional linear space, azimuth, scale and some projection parameters. So we need to use somehow classical 3D space. Idea is to split our bizzare space into parts which separately are vector spaces. Position of game object is now information about concrete part of the world and coordinate vector from vector space of the particular part. Now is clear why we can on lower levels of abstraction use only classical linear space too.

Back to the game objects once again. Game object provides basic info about everything in the game. It’s also provides projection into quadrangles: takes vector space coordinates from position, size and azimuth and calculates four corners of the rectangle. Now we can implement many of game object derivations: spatial objects carrying 3D model or flat ground objects and plates carrying only texture instead the 3D model.

We need decide how to split our bizzare space into mentioned parts. We make it by definition of the game. We split it by individual quarters. Town quarter is subset of linear space and this division will show up helpful in further game logic implementations.

From the above description it follows that we can correctly compute collision between two game objects only if they are in the same quarter. It isn’t problem at all. All we must do is conceive game objects and game logic…………..

## Town generator

## Boxes, tools and action objects

## People, reflexes and tasks

## Opponent, task planning

## Player and game controls

## Settings and menus

# User documentation

# Bibliography

1. http://msdn.microsoft.com/en-us/library/bb401006.aspx