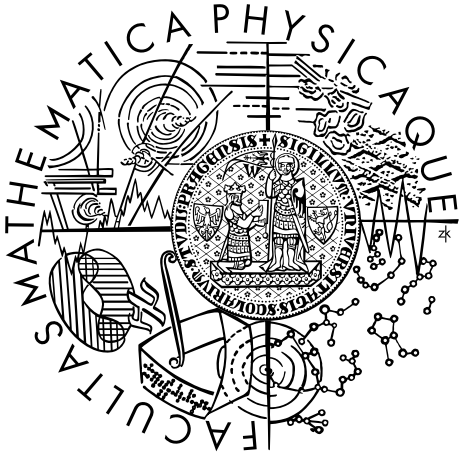
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Jaroslav Jindrák

The Dungeon Throne: A 3D Dungeon Management Game

Department of Distributed and Dependable Systems

Supervisor of the bachelor thesis:	Mgr. Pavel Ježek, Ph.D
Study programme:	Computer Science
Study branch:	Programming and Software Systems

Prague 2016

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 Sb., 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 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: The Dungeon Throne: A 3D Dungeon Management Game

Author: Jaroslav Jindrák

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D, Department of Distributed and Dependable Systems

Abstract: Abstract. TODO:

Keywords: dungeon management Lua scripting Ogre CEGUI

Dedication. TODO:

Contents

1. Introduction

Until the release of Dungeon Keeper[1] most well known fantasy video games have allowed the player to play as various heroic characters, raiding dungeons filled with evil forces in order to aquire treasures and fame. In Dungeon Keeper, however, we join the opposite faction and try to defend our own dungeon (along with all the treasures hidden in it) from endless hordes of heroes trying to pillage our domain. Although we can still play the original Dungeon Keeper today, we cannot change its data or game mechanics in any easy way so this thesis aims to recreate and modify the original game and extend it to have an easy to use programming interface that will allow such modifications.

1.1 Dungeon Managment Genre

Dungeon Keeper was the first game released in the dungeon management (DM) genre and since our game is going to be based on Dungeon Keeper, we should design it with the elements of its genre in mind. Since the definition of this genre has to our knowledge never been formally documented by the creators of Dungeon Keeper, we are going to create a list of basic elements of the genre based on the gameplay of the original game.

In Dungeon Keeper, the player's main goal is to build and protect his own base, called the dungeon. They do so by commanding their underlings (often called minions), whom they can command to mine gold, which is a resource they use in order to build new rooms and cast spells (in the game's sequel[2], mana was added into the game as a secondary resource used for spell casting), which could be researched as the game progressed. They would then use creatures spawned in their buildings as well as their own magic powers to fight intruders in order to protect their dungeon. From this brief gameplay summary, we can create list of the most basic design elements, which can be found in dungeon management games:

(E1) Resource management

(E2) Dungeon building

(E3) Minion commanding

(E4) Combat

(E5) Player participation in combat

(E6) Research

Now that we have created a list of the genre's basic design elements, let us have a look at how they were implemented in games from the Dungeon Keeper series.

[1] Bullfrog Productions, 1997

[2] Dungeon Keeper 2, Bullfrog Productions, 1999

Resource management

In the original Dungeon Keeper, the player used gold as their primary resource. They would have it mined by their minions and use it to build new rooms and cast spells. While having a single resource for everything may bring simplicity to the game, it also means that once the player runs out of gold, they wont be able to directly participate in combat due to their inability to cast spells. It is for this reason that we are going to use the resource model of Dungeon Keeper 2 and have a separate resource – mana – that will be used to cast spells.

Ask PJ: Should the word mana be itallic in the same way mods and modding tools are? This is its second appearance in the text, but the first was in a footnote.

Dungeon building

The term dungeon building in Dungeon Keeper refers to tearing down walls in order to create new rooms which the player's minions then claim and the player can place new buildings in. These buildings can then act as gold storage, spawn new creatures or be used as traps that negatively affect attacking enemies. Since this is a central theme in Dungeon Keeper (and dungeon management games in general), we should try to implement our building model to resemble this.

Minion commanding

Combat

In Dungeon Keeper, the player's dungeon is under frequent attacks by enemy heroes. These enemies attack the dungeon in groups with a delay between each two attacks, in a similar way to tower defense games (e.g. Orcs must die! [1] and Dungeon Defenders [2]), where theay are often referred to as *waves*. Each wave of enemies can consist of different types of heroes and the delay between ways can differ, too. Since the players of the original Dungeon Keeper are alredy familiar with this wave system and it also caters to players of tower defense games, we will be implementing it in our game.

Player participation in combat

During the fights with enemy heroes, the player in Dungeon Keeper can cast various spells to affect the outcome of the battle. These spells can tave various forms from spawning creatures, damaging enemies and healing minions to destroying walls and throwing meteors.

Research

Conclusion

TODO: Conclusion.
The game has to be a full competetive product, not a prototype.
It has to be performant, achieving high framerate even on low end computers.
It has to offer full single player experience, with scripted enemies and a chance to both win and lose.

1.2 Modifiability in Games

One of our basic goals is for our game to be modifiable, which means to provide tools – often called *modding tools* – to our players that will allow them to create modifications – often called *mods* – that other players can install and which can change or add elements to the game.

This can increase the replay value of the game as after finishing it, more missions, characters, game mechanics, abilities, items or even game modes can be easily downloaded and installed from internet. Since we want our game to be modifiable, we should allow our players to change most (if not all) of the game's data, artificial intelligence and abilities of all minions, creatures and enemies and also give them the ability to edit saved states of the game, which would allow them to create new maps.

Modding tools can generally be of two types: (1) an editor (like a world editor in Warcraft 3 [3] and Starcraft [4]) or (2) an application programming interface (API) that can be used in a scripting language (used in e.g. Starbound [5] allows its players to create modification either by editing JSON[3] files or by using its Lua[4] API).

(1) While an editor that would allow the players to change the game's data and its save files would be easier to use due to its point and click nature, the development of such tool is unfortunately beyond the scope of this thesis and would probably have to rely on an API (for code generation) anyway. For these two reasons this option won't be used, but creates a possible future extension of the game, which would extend the modifiable part of the game to even bigger audiences.

(2) Creating an API that can be used in a scripting language (e.g. Lua) may, besides making the game modifiable, lead to faster development if the main part of the game is written in a compiled language as parts of the game that are not performance critical can be written in a higher level, dynamically typed and interpreted language with an easy to understand syntax. These characteristics of scripting languages can make mod creation easier to non-programmers [8] and support rapid iteration [9] (as they do not require recompilation) and are the reasong why we are going to use this option.

Now that we have decided on the technical aspect of the modding support in our game, we need to decide which parts of the game we will allow the players to modify. An example of an easily modifiable game is Minecraft [10], a 3D sandbox game in which the player has to survive in a procedurally generated world. They build a house, craft tools, mine for materials and fight off enemies during the night. As we can see the game is very data oriented, with different crafting recipes, enemies, blocks to build with and ores to mine being the main part of the gameplay because the actual goal (to survive) is quite simple. This allows the mod creators to possibly prolong the time spent with the game by several hours (per mod) by adding these new items or by creating new custom maps that offer new challenges to the player.

Aside from adding items and entities, entire new games can be created within

[3]JavaScript Object Notation, a lightweight data-interchange format designed to be easy for humans to read and write while being easily parseable by computers.[6]

[4]A fast, lightweight and embeddable scripting language.[7]

a modifiable game. In an interview on the server PCGamer [11] the lead developer of Minecraft, Jens Bergensten, said: *"In the recent snapshot we've added something called the command block. When it's triggered, you can teleport players to a certain position, or change the game mode. So this user called Sethbling has created a Team Fortress 2 map for Minecraft. You can choose a class, it has control points, and everything works like TF2. It's quite amazing."* (More information about Team Fortress 2 can be found at its official website [12])

To allow the creation of custom maps in our game, we should make a save feature which will create levels in a format that will allow later modification of the saved state of the game, including actual changes to its gameplay.

In conclusion, we can see that the ability to modify a game can help said game to grow even when its development has stopped or is focused in different areas (e.g. security, stability). Since we want to give this ability to the players of our game, our modding API should allow them to change most of its data, including minions, enemies, buildings, spells, artificial intelligence and others, but the game on its own should be fully featured, offering enough of these entities on its own so the players do not need mods to actually play the game. Additionally, since our game (like Dungeon Keeper) will have scripted waves of enemies attacking the player's dungeon, the also should allow our players to alter the wave composition and delays. Last, but not least, we must not forget that players do not necessarily have to be (and often are not) programmers, so our game should provide an easy way to install these modifications.

TODO: Ask PJ: I talk about gameplay, should I explain the term (oxford dict definition, could be in a footer for example)?
(http://www.oxforddictionaries.com/definition/english/gameplay;
"The features of a computer game, such as its plot and the way it is played, as distinct from the graphics and sound effects.")

1.3 Thesis Goals

The main goal of this thesis is to design and implement a modifiable 3D dungeon management game using the design elements **(E1)** – **(E6)**.

In addition to the main goal, the game should complete the following list of goals:

(G1) The game has to be a full competetive product, not a prototype.

- **(G1.1)** It has to be performant, achieving high framerate even on low end computers.
- **(G1.2)** It has to offer full single player experience, with scripted enemies and a chance to both win and lose.
- **(G1.3)** X It has to contain a variety of entities, spells and buildings even without mods.

(G2) X The game has to be highly modifiable, providing an easy to use modding interface for players.

- **(G2.1)** X The mod creators must be able to create new entities, spells and buildings and to change most of the game's data.
- **(G2.2)** X They must also be able to alter the game progression by defining enemies that spawn and delays between them.
- **(G2.3)** X The game has to have a save feature which allows the mod creators to change the save files to create custom levels.

(G3) X The mods for the game have to be easily installable even by players without any programming knowledge.

Bibliography

[1] Orcs must die! http://en.omd.gameforge.com/. [Online; accessed 2016-05-15].

[2] Dungeon Defenders. https://dungeondefenders.com/1/. [Online; accessed 2016-05-15].

[3] Warcraft 3. http://us.blizzard.com/en-us/games/war3/. [Online; accessed 2016-05-09].

[4] Starcraft. http://us.blizzard.com/en-us/games/sc/. [Online; accessed 2016-05-09].

[5] Starbound. http://www.playstarbound.com. [Online; accessed 2016-05-09].

[6] Javascript object notation. http://www.json.org. [Online; accessed 2016-05-09].

[7] Lua. http://www.lua.org. [Online; accessed 2016-05-09].

[8] Garage Games. Why Use Scripting, Torque Engine documentation. http://docs.garagegames.com/tgea/official/content/documentation/Scripting%20Reference/Introduction/Why%20Use%20Scripting.html. [Online; accessed 2016-05-10].

[9] J. Gregory. *Game Engine Architecture*. 1st Edition. A K Peters/CRC Press, Boca Raton, FL, 2009.

[10] Minecraft. http://www.minecraft.net. [Online; accessed 2016-04-23].

[11] PCGamer. Future of Minecraft. http://www.pcgamer.com/the-future-of-minecraft/, 2012. [Online; accessed 2016-04-23].

[12] Team Fortress 2. http://www.teamfortress.com. [Online; accessed 2016-04-23].

Attachments