AALBORG UNIVERSITY
STUDENT REPORT

2D GAME DEVELOPMENT

DIMEY

THE GROUP 7
ITCOM

DECEMBER 19TH 2016

Project in Object Oriented Programming, P1-project

Supervisior
Lene Tolstrup Sørensen

Technical supervisior
Sokol Kosta

Authors
20163989 Pawel Kowalinski
20164637 Aleksander Nowak
20165335 Maciej Krygier

Pages
32

Table of content

Introduction

Video games have vast impact on modern society, affecting our lives in various ways. They fill up our free time, letting us experience imaginary worlds. Developers push their skills to the limits with every new game they create. That's why, with every year we receive more beautiful and breath-taking titles. Todd Howard and his team of a hundred people created *Fallout 4,* a 2015's Game of The Year Nominee[1]. In an interview for Gamespot, he says "We're probably doing too much" and explains problems that he faced during the development of such an advance technologically title[2]. On the other side of the coin, independent studios made of only a few video game developers also produce original titles, focused around gameplay and atmosphere. Those games do not fascinate with extraordinary visuals, but with original ideas, brave solutions and exceptional artworks. Games like these are popularly referred to as indie games and quite an amount of them appeared to be a huge success. Examples are the following: *Enter the Gungeon, Nuclear, Stardew Valley,* or *Hyper Light Drifter* (last 2 nominated on Game Awards 2016 as "Best Independent Game")[3].

Motivation & Problem formulation

We all play games and we are naturally interested in this industry. While deciding on our project goal, we were thinking of something which would be challenging, yet fun and interesting for us to make. We also wanted for the game to be original and use unconventional ideas to keep the player invested into it. Finally, we decided on making a pixel-art 2D game in which a player would be able to access a 3D world. Therefore, we formulate our problem as follows:

How to develop pixel-art 2D game?

- How to present 3D scenery in 2D game?

General Research and Methodology Characteristics

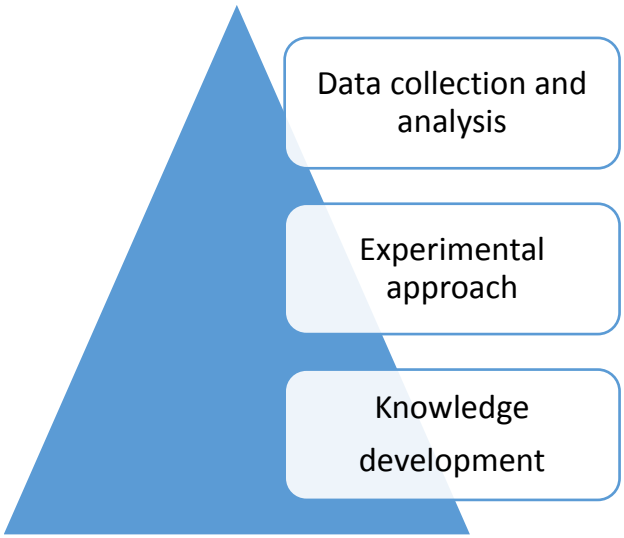

Figure 1
Game Development Knowledge

At the start of our project we had no experience nor even knowledge about the game development. After several discussions and chaotic experiments our general idea of methodology evolved into *Figure 1* shape which represents a process of knowledge understanding and acquisition. The chart is based on "The research 'Onion'" diagram[4]. This deductive mindset guided us through learning process and resulted in functional game application.

The prototype

Our first step to initiate the project was to organize our project work scheme – the things the group is supposed to accomplish in order to finish the project successfully. We had to come up with the starting point – a basic level of knowledge each member of the group should move toward and take it as our initial goal. As a result, the team was able to efficiently co-operate in game development world, avoiding knowledge gaps and miscommunication issues. The idea of the starting point expanded into objective of building a prototype software by each member independently. The prototype would contain a window application with a movable object on the middle. To successfully complete that part of the project, it was required to conduct an extensive research with bustling brainstorms and common investigation described as data collection and data analysis respectively.

Data Collection

The desired information was the data about libraries, frameworks and overall design patterns to be found in game development. Our data sources consisted mostly of the internet resources wrapped around game development forums, GitHub repositories and the most popular search engine's records. Eventually, the most significant research content we managed to gather accumulated in three main sources – straight-forward online programming tutorials in a form of articles or videos, documentation of researched libraries or programming language and the "3D Game Development with LWJGL 3" online book published by GitBook member called "lwjglgamedev".

All of the obtained data have to be processed by researcher's analysis and consultation with the group. This will provide conclusions which may lead to increasing of our overall knowledge about game development, ultimately producing results in the project itself. Structure of analysis will be described in two following paragraphs.

Passive data analysis

In our case data analysis is divided into two sections – active and passive. The passive way means that the researcher does not interfere with obtained data – meaning they try not to contaminate the data with their own interpretations and usage.

The researched content goes through general, overall analysis with regards to three factors. First are brief descriptions provided by authors or publishers. In this part we gain, for example, specifications of technology like a name or its general purpose. At this level we can decide whether the technology is suitable for our goals (i.e. in terms of operation system requirements). The second factor is an example of implementation of given technology, where we can see in what way it has been used in real life production. We can approach this by reading the code or documentation of already existing products. Thus it shows us the technology management by more experienced developers. The last factor is our overall view or an opinion about a final product developed with researched technology by some company, thus predicting our future results and deciding whether or not they fit the project goals.

Active data analysis – Experimental approach

Finally, we can choose the most appropriate record and use it in an experimental fashion. This is the active side of data analysis where the researcher is meant to actually use collected data on its own. It is a main difference between active and passive approaches.

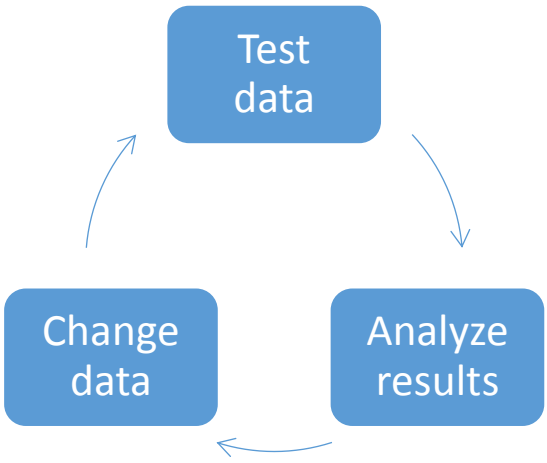

Figure 2
The Processing Cycle

To begin with an active analysis, we need prepared data and local environment – an operational system with compatible IDE (Integrated development environment). Then, the developer will go through undefined number of iterations of the Processing Cycle until the satisfactory extend of understanding of collected data has been reached.

Experimental approach – an example

To illustrate general idea of this method we will use an example. Let say that we want to create desktop application having certain height and width of its window. Following the Figure 2, one of iterations may look as follows. Firstly, we need to find a code which would suit our goal, so we do some research, then we run the code on our IDE ("Test data"). Secondly, we have to determine whether the code gives desired results ("Analyze data"). Finally, we are able to create window with certain size, then we can try figuring out how change the window into Full Screen mode on our own ("Change data").

To sum up the example – goal of the application in certain height and width resulted in the application in full screen mode. This additional knowledge that we obtained is called empirical knowledge and is based on our experience.

Conclusion

Gathered data combined with continuous preliminary programming finally succeeded into previously mentioned prototype. Eventually, we have obtained and developed our own knowledge which can be used as a tool to create, expand and manipulate built software. Furthermore, we can compare our empirical knowledge with authorities' materials to distinguish dependencies and extract even more relevant observations. This creates an opportunity to discover deep understanding on both practical and theoretical levels.

State of the Art

Indie game is commonly known shortcut from “independent video game”. By the definition, it means a game which was created without a financial support of a publisher OR without directional influence of a publisher during the development[5]. Indie games are usually created by small teams with a singular project goal of creating the game. One of the most relevant advantages of indie developers is the independence in creating the title. Publishers tend to impose their limitations in terms of creating a game due to influence of investors being more interested in the financial success of the game rather than the vision behind the original idea. Also with crowdfunding services (kickstarter.com, gofundme.com, Indiegogo.com) being popular these days, developers can raise budget needed to create their game[6]. A lot of online platforms like Steam or gog.com allow independent developers to publish their games as they see fit, which was not possible in the past, since games were distributed mainly in physical form then and crowdfunding didn’t exist.

What makes indie games attractive for customers? They often don’t bring the same graphical quality as full-priced games from popular producers like Electronic Arts, Ubisoft, or Bethesda Softworks, simply because of the lower project budget[7]. Indie games focus more on exceptional and inspiring stories, interesting gameplay mechanics and original artworks. One of the most popular art style among indie games is pixel-art. Most popular indie games of 2016 were created in this artwork, for example: “Enter the Gungeon”, “Hyper Light Drifter”, “Nuclear Throne” and “Stardew Valley”[8]. What is more, one of the most recognized games of last decade, Minecraft, was also created in pixel-art and developed as an indie game [9]. Those games provide challenges to players, backed up with beautifully designed worlds, and other original approaches to video gaming. Indie developers primarily design 2 dimensional games, since the development of those games is far easier than 3D games. Programmers discuss this subject on various forums generally consider 2D development as easier and less complex than 3D[10].

We analyzed other titles, focusing on how they present 3D in 2D world. We played and analyzed every game listed below, focusing on visual representations of depth. As a product of this analysis we obtained several techniques of presenting 2D scenes.

Hyper Light Drifter

Figure 3
In-game screenshot

Hyper Light Drifter is a game created by Heart Machine studio, directed by Alex Preston - who came up with the idea for a whole game. Preston was born with congenital heart disease and he wanted present his experiences through a video game. The game was created with "Game Maker" software designed for game development.[11]

HLD utilizes techniques which are informally called pseudo–3D. The scene is presented under an angle in regards to the player, which allows presenting distant background layers and plans. To achieve the depth of those scenes, it uses different **layer scaling** and **parallax scrolling** to show us the distances between player and the background. Parallax scrolling is a technique where background layers move slower than the foreground layers which gives us the simulation of depth[12]. When we look at smaller operations to achieve three-dimensionality we can extinguish light and shadows play and slowing movement speed while climbing in the environment like stairs. After consideration and observations about different perspectives and game scene presentations, we can conclude that HLD is a 2D game, which uses graphical projections and techniques to fake 3D.

By analyzing this game under development aspects, we learned how to use camera angle to imitate 3D. We also got to know different scenery presentation techniques like layer scaling or parallax scrolling.

Titan Souls

Figure 4
In-game screenshot

Next game on our list is a production of Acid Nerve – Titan Souls. It's a top-down action based game where we incarnate as our nameless hero, equipped only with a bow and an arrow. This title is known for its difficulty, which draws from popular series *Dark Souls*[13].

Again, through our observations we can tell, that Titan Souls uses similar techniques as Hyper Light Drifter, like using similar camera angle to expose further layers of the screens, working with light and shadows. In particular, Titan souls utilizes **multi-layering**, which means rendering different layers of scenes on top of each other. This is connected with collision detection, and for example, if we approach a tree in the game from behind, we will be able to move *behind* its texture, which gives us the imitation of third dimension. This game also occasionally uses 3D models for boss' characters, like presented in a picture below.

Thanks to *Titan Souls* we learned how to use collision detection and multi-layering in order to imitate 3D world. We also realized that it is possible to mix both 2D maps and sprites with proper 3D models.

FEZ

Figure 5
In-game screenshot

FEZ is an indie puzzle-platform video game developed by Polytron Corporation and published by Trapdoor. We play as Gomez who receives a magical fez, revealing that his 2-dimensional world is only a side of a four-sided 3D world. Player faces multiple platform puzzles during which he must operate between those four views to solve them.[14]

Game approach to presenting 3D world is truly unconventional. A whole world is created with 3D models, but player can operate with only 1 side of those models at a time. That operation restricts the gameplay to the 2D perspective only, but player can freely switch between 4 perspectives. Besides that, it also uses multi-layering and scaling to present far objects in the scene.

During this research, we tried to find different ways to present 3D world in our project. We have learnt about types of 2D perspectives like top-down or side scrolling. We also got to know the techniques, that try to fake third dimension in a 2D world, like multi-layering, scaling and parallax scrolling. Even though, we didn't use some of those techniques to represent 3D in our demo, we expanded our knowledge in that area, allowing us to plan ahead certain functionalities. The example of technique that we would love to implement in the future would be parallax scrolling, especially in presenting height.

About the game

Dimey is a game telling a story about our main character – Dimey, who is a pixel living in a 2D world. The environment in Dimey is purely two-dimensional – characters in the game grew up in this world, they think in 2D, they live in 2D and they feel in 2D. Just like us, humans, just one dimension away. This is what makes our hero different from others. He has an ability to interact into depth, which allows him to experience his surroundings on a whole another surface. This is something surreal for other pixels and they will react differently to his actions. As a player we will experience being an entity almost alien to the rest of the universe. We are yet to see if we will be accepted, or pushed away into exile caused by the fear of the unknown.

Imagination plays a huge role in our game since players will be invested into exploring the world, figuring out the story and the role we are yet to play through dialogues and objects found in the game. The player will be given simple quests, but in addition to the 3D twist to them, their completion might bring surprising and interesting results. For example, a going through complicated maze does not stand a challenge for us when we can jump above the walls.

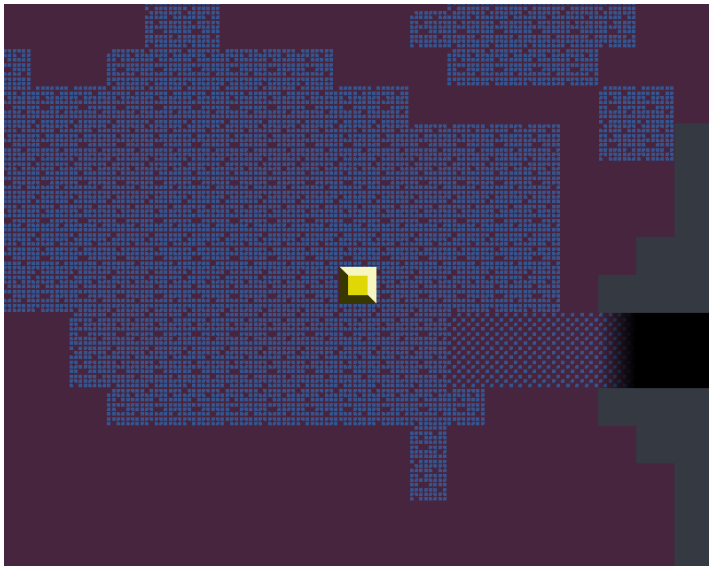

Figure 6
In-game screen

About the demo

We prepared a tech-demo of *Dimey* for the P1 project, which consists of a 3-level world the player can explore: a forest area, castle's corridor and castle's room. We enable changing dimensions from 2D to 3D and back to present our preliminary idea of how the dimensional system would work. We also designed a little splash art for the starting screen presenting key bindings used in the game.

Game Design

Dimey from the design perspective is the set of related objects. Every one of them contains meta information that user can perceive, interpret and process to receive some kind of context.

To create visual side of the game we had to accomplish certain chain of tasks, beginning with creation of elementary elements in 2D game design called tiles. Then, we have been able to merge all the tiles into a single set of tiles – which we used to create tile layers. Finally, we could bring together all of the layers into one functional tile map, ready to be implemented into the game. Below we describe a full list of components we have used, as well as the type of data they represent.

Tile

Elementary unit of our digital graphics is a tile in a shape of a square having 64 pixel long sides. We kept all tiles in PNG (Portable Network Graphics) format to support transparency (which is essential in Light and Shadow concept described later). All of the tiles are self-made, using GIMP software (GNU Image Manipulator Program).

The size of tiles used in 2D projects is not fixed and depends on the design approach. Initially, we planned to use 32px tiles. However, we noticed that the bigger the tile is, the more details are noticeable on the screen – choosing 64px tiles gave our graphics a more expressive look. All the patterns are drawn with squares or their imitations which simulate the pixel style of a game.

The tile may represent a pattern, solid color or a single element. Figure 1 shows principles from Dimey's implementation.

Figure 7
Tiles, in-use examples. Respectively: surface pattern, solid color of the void and the starting point of the 3D wall.

Tile layer

Tile Layer is a group of tiles put in aesthetical order to imitate image of the game's world. It is created using a set of tiles (the tiles merged into one file as shown in figure 3) to draw on an empty canvas using one tiles as brushes. Empty project layer is always divided into squares of the tiles size denoted as cells, so it simplifies the drawing. Technically, our layers have 1600px x 1200px size which implies 25 x 20 tiles of 64px long side.

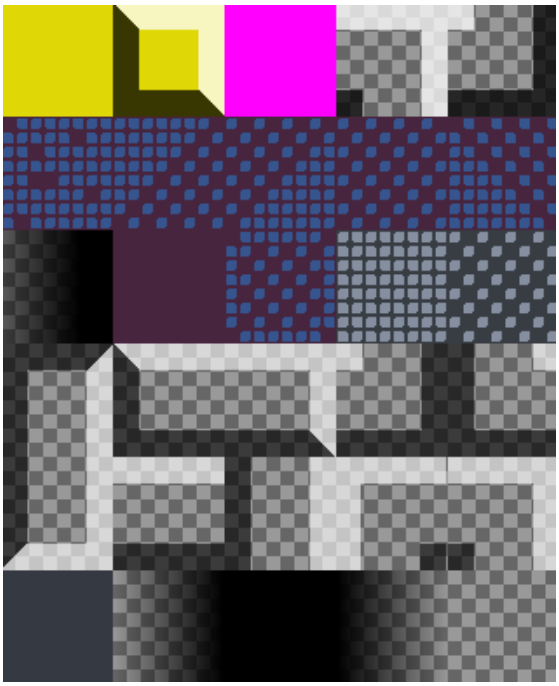

Figure 8
Tileset

Tile Map

Tile map contains several tile layers and objects put on top of each other, creating a visual map. Tiles and layers can be assigned custom properties and values of different types to distinguish, for example, specific types of surface.

Tiled

In order to create tile maps we used dedicated editing software called Tiled.[15] Maps created using Tiled software can be loaded into libGDX projects, which fully supports development using such maps. We are presented a variety of methods allowing for rendering, changing and manipulating TiledMaps. We took advantage of it by specifying which tiles are meant to block players movement and which represent portal to another location. Every collideable tile in our demo has a property "Blocked!", while portals have different values of their "Portal" property, passing an information of portal's destination.

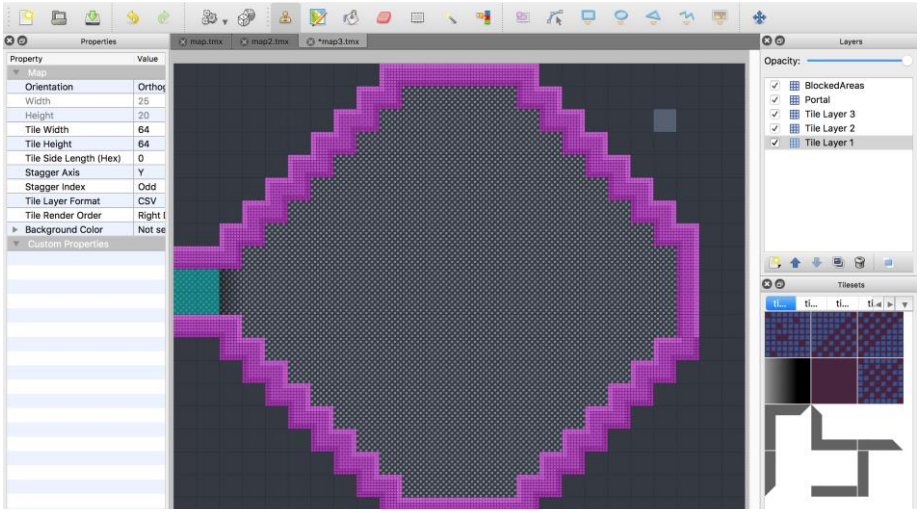

Figure 9
Castle room seen in Tiled. Collision tiles(pink), Portal tiles(tide).

In process of creating tiles we have used variety of color pallets to distinguish areas. Moving from one place to another imitates journey which affects game atmosphere. To signify that the tile before the portal is black-to-transparent gradient color to simulate the depth, absorbing darkness, an entry into unknown. Additionally, solid colors are not meant to be traversable by the character.

Sprite

Inheriting all the graphical properties of a tile, sprite has one crucial difference which is its ability to move around the screen. Thus, the main moveable character in Dimey is represented by a sprite. Figure 4 shows in-game instances.

Figure 10
Sprites, in-game examples

Camera

In Dimey, the camera scope does change every level. The further camera distance is, the greater area it covers. That is yet another technique, which imitates depth by showing scales of the locations. Camera in the demo constantly follows the player's sprite. Although this solution might cause motion sickness (for which we deeply apologize), the concept of associating the camera with player's exploration strengthen the idea that Dimey is purely a main character.

On the other hand, camera's perspective is set to top-down, which almost completely rejects third dimension and forces the character to move on two-dimensional plane. This solution puts an emphasis on the fact that the game's world has only two dimensions of height and width – thus traversing the depth would feel much more unnatural.

Light and Shadow

It is our main concept which helps representing a three-dimensional world in two-dimensional game. Technically, it lightens up a color of a tile on its top and right sides and darken on the opposite. It is based on an imitation of the sun placed somewhere in a top-right corner of game's world. Thus, the imaginary sunbeams fall off on the objects in game creating optic illusion which makes them look like a solid figure with its own plane base and height.

Creation process of all graphic components connected with 3D mode was highly influenced by the Light and Shadow concept. For example, wall tiles had one additional layer on top with transparency set on 75%. As a result, solid white and black colors looked more similar to a shine and shadow respectively. It helped to make the depth effect more realistic and immersive.

Splash art

The initial graphical component that player sees immediately after launch is a splash art. It is a widow-sized image with the title and basic controls drawn on top of it. Splash provides an introduction and guidance into Dimey, making the the player familiar with the game and creates an appealing first impression. The splash screen would be replaced with a proper menu in the future.

Game development research

A video game project presents a wide variety of programming problems when speaking about its technical side. The solutions to these problems are not always straight forward, and the tools utilized to solve them require an immense amount of time spend on educating programmers, until they reach a satisfactory extend of familiarity. Only then it is possible to effectively and efficiently create robust and solid software following the vision of team undertaking such a challenge. Since in recent years the Internet has become the most common source of information[16], and on the other hand software development is one of the most valuable skills in the industry[17], internet became a go-to source of information regarding video game development. That being said, one can google-up a tutorial to almost anything, but a wild nature of the Web saturates academic and industry-standard educational papers from general how-to cookbooks, which are mostly written by unchecked authors. It creates a need of searching for the quality educational materials and choosing right tools in regards to the scope of the project and the time-frame in which the software is being developed.

Creating Dimey was a process of learning know-how of the video game development, and choosing the right means to do so. Research was done on three primary tools being used the most often amongst Java tutorials on “How to program a game” :

- Swing
- Lightweight Java Game Library
- libGDX

Swing

Built into Java Developer Kit libraries allow to create simple window applications in Java without adding any external tools. In 1996 Java released library called AWT[18], which provides Java programs with a GUI (graphical user interface). Original destination of AWT, later updated with Swing, was to allow programmers from many different industries create simple window applications which could run on any device supporting Java Virtual Machine[19]. Swing inherited the previous nature of AWT, making it tough for game developers to get to the functions that they really need. The vast amount of different methods and their overloads is overwhelming. Thus developing such refined software such as games using Swing requires relatively high level of initial understanding and familiarity with Java native libraries just to get started.

```
// JFrame window setup
public GameCwiczonka() throws IOException {

    setMinimumSize(new Dimension(WIDTH * SCALE, HEIGHT * SCALE));
    setMaximumSize(new Dimension(WIDTH * SCALE, HEIGHT * SCALE));
    setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT * SCALE));

    // Create an instance of JFrame
    frame = new JFrame(NAME);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // Laying out the JFrame
    frame.setLayout(new BorderLayout());
    // Center Canvas in the center of JFrame
    frame.add(this, BorderLayout.CENTER);

    frame.pack();

    frame.setResizable(false);
    frame.setLocationRelativeTo(null);
    // Make the window ACTUALLY visible
    frame.setVisible(true);
}
```

Figure 10
Example block creating window using Swing.

However, learning the game development from scratch allowed us to understand the base principles to the video game coding, such as the game loops, the render functions, input handlers, and even more sophisticated topics such as thread manipulations. Having this basic understanding, we could direct our research into more advanced educational materials, which ultimately lead us to understand how games generally work. We used this knowledge in our State of the Art, trying to analyze popular games in search of avenues for presenting three-dimensional scenery using 2D components.

Lightweight Java Game Library

LWJGL is an open source Java library, enabling access to some of the popular game APIs[20]. Two of those APIs are OpenGL (for processing graphics) and OpenAL (for audio), of which we knew that they were being widely used in 2D game development. Using specialized library for the game development sounded really attractive to us, and for some time we decided to make it our final tool choice for this project. To gain familiarity with the tool, in addition to many internet tutorials and the documentation, we used "3D Game Development with LWJGL" book made by a GitBook user who gathered a lot of information from the Internet, segregated it and preserved in a form of the book[21]. After a while of learning LWJGL, we discovered that as promising as it was, the library did not operate with the higher-level utilities than what the native libraries expose[20]. What it meant for our project was that although the library is near perfect for the rapid development of the game, it requires a humongous amount of time to learn its functionalities. Ultimately, it lead us to using unintuitive or hard-to-understand method signatures, which behavior is hidden behind the implementation of the library, and the documentation provides even more questions than answers. Covering a wide range of topics, from the simple window creation and set-up, to rendering custom shaders, learning LWJGL ended up being just too complex and time-consuming for the P1 time-frame. Even the developers of LWJGL suggest that using their library by beginners is a challenge, and recommend finding framework that implements LWJGL library[20].

```
    // Create new window
    int WIDTH = 300;
    int HEIGHT = 300;
    window = glfwCreateWindow(WIDTH, HEIGHT, "New Game", NULL, NULL);

    // Get the resolution of the primary monitor
    GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());

    // Set up windows position
    glfwSetWindowPos(
            window,
            (vidmode.width() - WIDTH) / 2,
            (vidmode.height() - HEIGHT) / 2);

    // Make the OpenGL context current for the thread
    glfwMakeContextCurrent(window);

    // Enable v-sync
    glfwSwapInterval(1);

    // Show the window
    glfwShowWindow(window);
}
```

Figure 11
Example block creating window using LWJGL.

We consider the time spend researching the library a good investment. Not only did it deepen our understanding of the video games development overall, it also increased our individual skills in programming. “How to develop 3D games with LWJGL” book gave us an insight into more advanced software development in general – the question “How to do it well?” started becoming more impactful to us rather than simple “How to do it?”. Author of the book himself often explained design choices, making reader consider the overall architecture and technological thought put into produced software, for example:

“The class hierarchy that we have created will help us separate our game engine code from the code of a specific game. Although it may seem unnecessary at this moment we need to isolate generic tasks that every game will use from the state logic, artwork and resources of a specific game in order to reuse our game engine.”[21]

From this fragment only we learned, that separation of the game engine from other components of the game makes it reusable across many different titles. The book also presents useful concept such as game logic, camera processing, rendering process in detail, and many more – all of these we used to some extend while creating our final game using libGDX framework.

LibGDX

LibGDX is a cross-platform game development framework. Following the developers vision, it provides an environment for rapid prototyping and fast iterations. It wraps up many popular game libraries including LWJGL, box2D, OpenGL/OpenAL and many more, into a simple Java API that is meant to help programmers with common game development tasks such as rendering scenes, playing sounds, building user interfaces and using trigonometry and linear algebra[23]. What is more, being a high-level API, it also allows to dig into lower-level functionalities of its components, meaning that depending on the necessity, we could either use the general and simple tools effectively, or utilize more specialized methods to develop complex functionalities as we progressed with the vision for the game. The documentation for libGDX consists of the set of tutorials on how to use different features, and the Javadoc explaining features in detail. This way we could not only start developing the game right off the bat, but also begin to discover other possibilities that libGDX presented us. The great advantage of libGDX over both LWJGL and Swing lies in its simplicity. Self-explanatory functions and easy to understand class structure allowed us to intuitively write fully operational code and immediately see its effects in debug mode. Additionally, the API was so easy to understand and operate that we could focus on improving the actual class structure of our program, rather that figuring out the ways to make these classes work in the first place.

```
class DesktopLauncher {
        public static void main (String[] arg) {
                LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
                new LwjglApplication(new MyGame(), config);
        }
}

class MyGame implements ApplicationListener {

    @Override
    public void create() {}

    @Override
    public void resize(int width, int height) {}

    @Override
    public void render() {}

    @Override
    public void pause() {}

    @Override
    public void resume() {}

    @Override
    public void dispose() {}
}
```

Figure 12
Example block creating window using libGDX.

Implementation

Given time-frame for the P1 project, we managed to gather information about different tools used in 2D game development. We also drastically increased our awareness of mindful software development in general. That being said, the product of last months is just a seed of the game, which we managed to developed during our research.

UML

UML diagram of the whole project can be found under this link:

https://drive.google.com/file/d/0B5n_9q1wMcPtRjFjWUpVN3ItZXM/view?usp=sharing

Core classes

LibGDX provides a simple wizard tool, which creates a project ready to be imported into desired IDE[24]. Straight out of the box we got two raw classes: GameCore and DesktopLauncher.

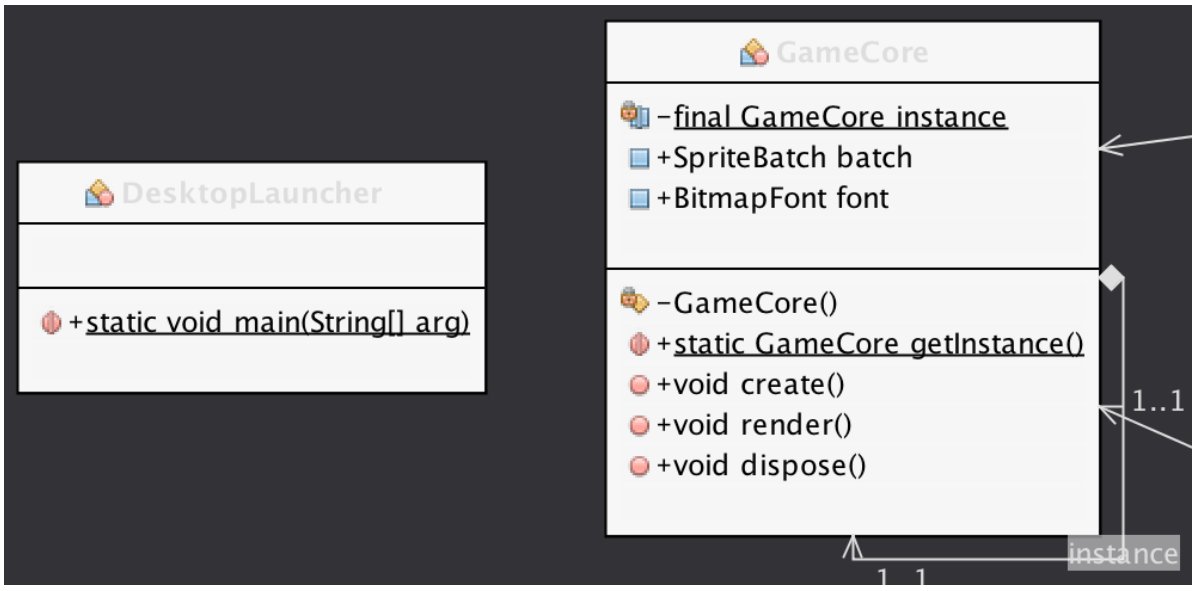

Figure 13
DesktopLauncher and GameCore classes

Both are already set up so the game could run a simple window with an example image. **DesktopLauncher** class is primarily used to launch the game and read some initial configuration, like the size of a game window, therefore it remained similar throughout our development. The implementation of the game begins execution with the **GameCore** class. Here we create the basic objects used for rendering (batch, font), load all the resources used by the game and - when all is done - send other game components a message to show the game. Being kind of a spine for the whole program, we made sure that only a singular instance of this class could exist at any given time, by applying eager instantiation creational design pattern over it[25].

Resources

Resources class provides content to be accessed statically for any game class. Creating this class resolved an issue of performance loss each time we were loading content to be rendered on a screen. In the early version of our program we noticed that each time we were, for instance, changing the currently shown map, the game would shortly freeze. Later on we figured out that files being loaded several times during the games execution were causing those freezes. To work around that, we decided to load all the content files before the game starts, store them in the reference variables and access only those we need at a time, which drastically improved games' performance.

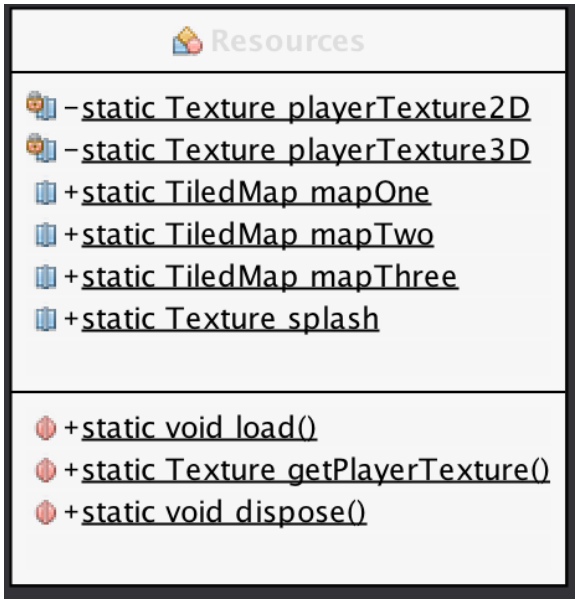

Figure 14
Resources class

Screen Management

Our game operates with Screens, which are essentially views being shown to the player. Screen interface from libGDX provides methods used to operate with those views. Each view is an independent instance of a Screen class, and there can be only one view existing at a time.

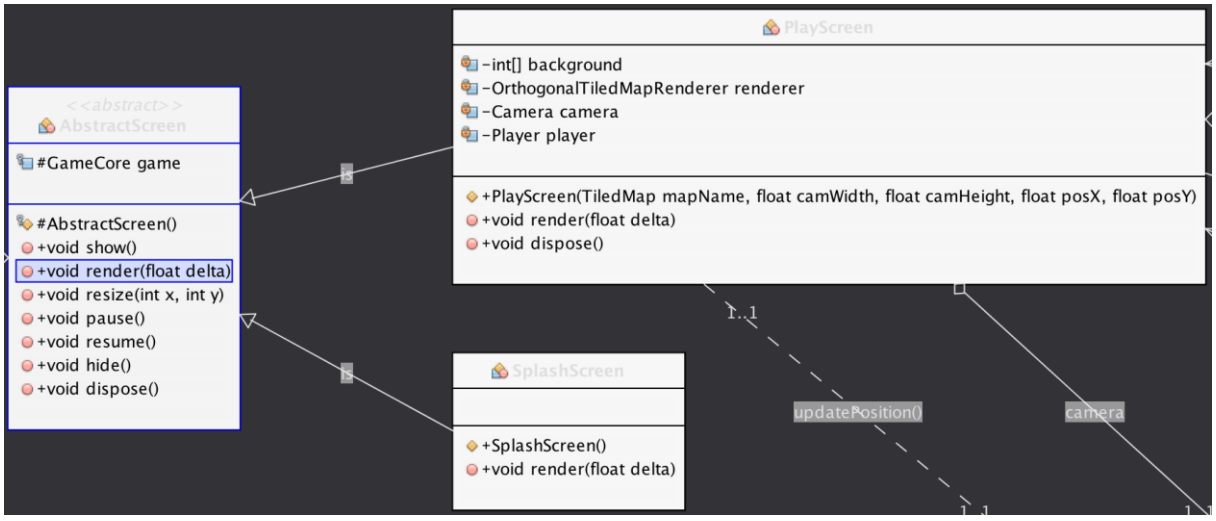

Figure 15
PlayScreen and SplashScreen are subclasses of AbstractScreen

Each type of view has its own class, where its specific behavior is defined, for example, the **SplashScreen** view has to only render a splash image and a message for the player, while the **PlayScreen** has to create specific objects for the game such as Player or the Camera (both will be explained shortly).

During the development we used to implement all the methods from the Screen interface into our views, even those we did not want to support. To avoid leaving empty methods in the classes and improve code readability, we decided to create our own type of Screen – **AbstractScreen**, which directly implements the Screen. All our views inherit from **AbstractScreen**, thus allowing us to override only those methods we want to support at a time. **AbstractScreen** also propagates the GameCore instance amongst all of the views, giving them the ability to use objects necessary for rendering. Being inspired by the article about Screen management from the PixNB Blog[26], we designed Screen Management system.

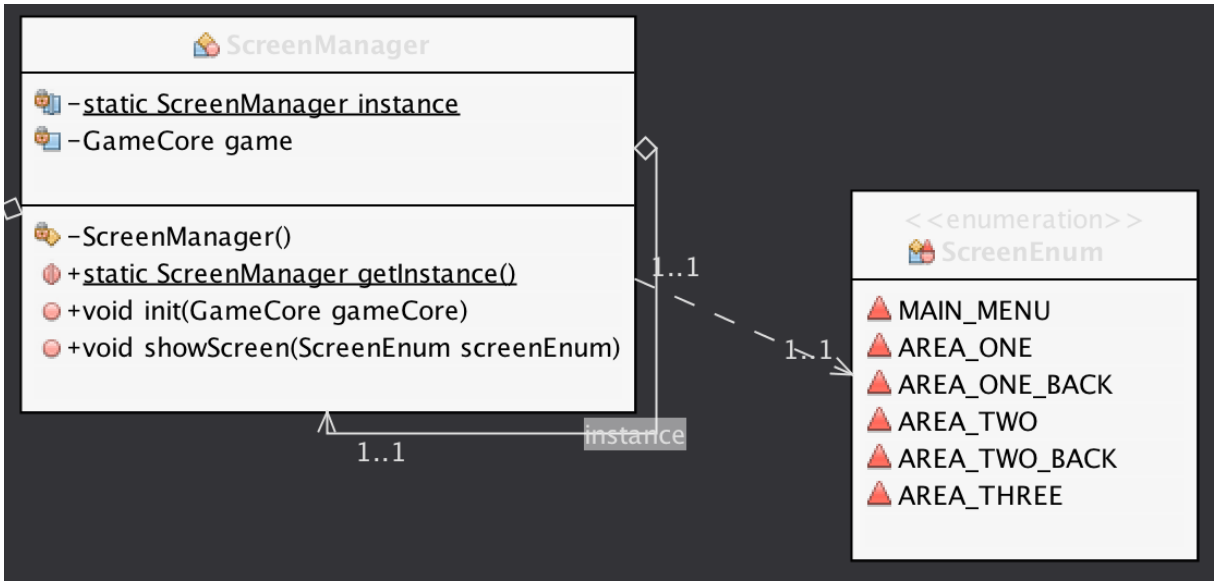

Figure 16
ScreenManager uses constants specified in ScreenEnum to change views.

It consists of a **ScreenManager** class and a **ScreenEnum**. In **ScreenEnum** we specify each view to be presented during execution of the game. **ScreenManager** is a singleton class that allows game to change current view into another one at any time. It also takes care of disposing the previous views – this way we make sure that our game has only one view presented at a time. The solution allowed us to effectively switch between different TiledMaps. Most importantly though, it is a very scalable approach - meaning it allows for a rapid development. Adding a whole new level to our game (no matter how big and complex the map would be) is a matter of less than ten lines of code.

Object classes (Player, Camera)

There are two other classes that were mentioned, but not explained. These classes represent objects that could be greatly improved both conceptually and programmatically, but unfortunately we had to take compromises due to the time limit of the P1 project.

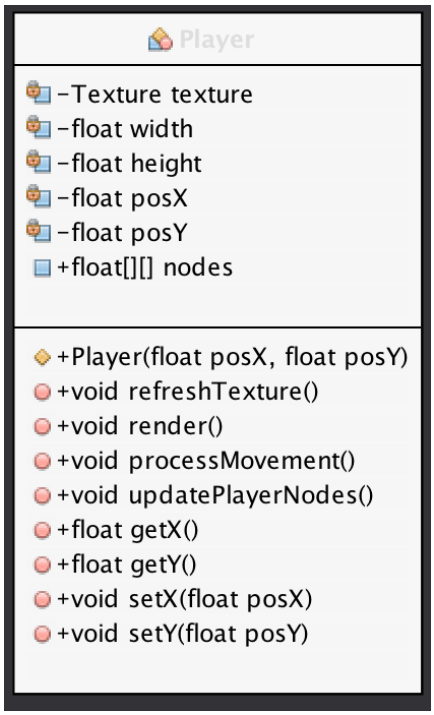

Figure 17
Player class

The **Player** class represents players' entity (including its position and attributes) – it also manages Players movement according to users input, and implements a system we called "nodes". We will explain this system shortly. Given more development time, we would try to separate movement from the Players class and improve it, to create movement as a feature suitable for other entities in the game (i.e. NPC). A fair idea would be to have vectors represent movement, instead of changing the position coordinates of a player. We would also try to distinguish input handling from the game engine – this would be beneficial if we decided to develop a touch handling for smartphones.

Camera class utilizes OrthraphicCamera from libGDX. It is a type of camera that provides only orthographic projection of the screen, meaning that it cannot present depth. Orthographic projection is meant to be used by simple 2D games, and is represented by two perpendicular lines which intersect at the middle of the viewport (an area of the screen that is being considered as a display for the game)[27]. In our prototype, we decided for the camera to be constantly changing its position so it matches the position of the Player.

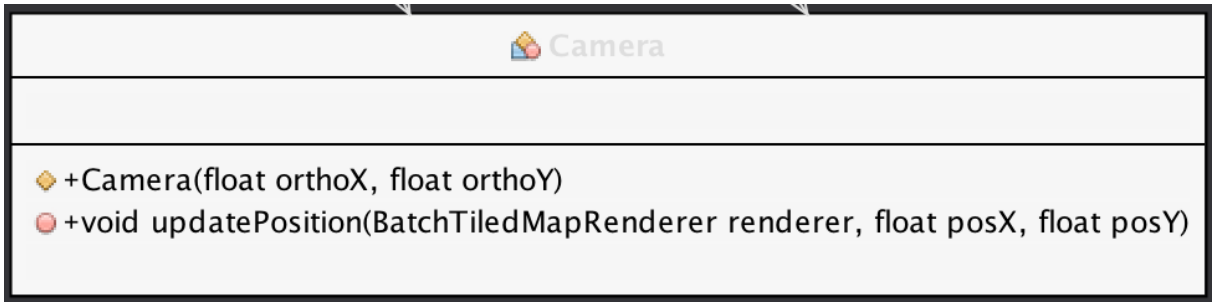

Figure 18
Camera class

However, most games that we investigated used refined cameras with complex geometry, therefore we would have to increase our knowledge on this topic in general to produce higher quality camera. That being said, the sort-term goal for us would be to make the camera limit its movement only to the rendered parts of the game – in our current version of Dimey, camera also covers non-rendered areas presenting a pitch-black zone around the map.

Detectors

We took an idea of creating single-instance classes and developed two singleton detectors. **Detectors** are classes that are being supplied with the TiledMap and the Player instance, to create the system that listens for a special events during the game (i.e. collision with a wall or the player entering another area). These classes have the ability to extract information about certain event they are meant to process from desired TiledMap.

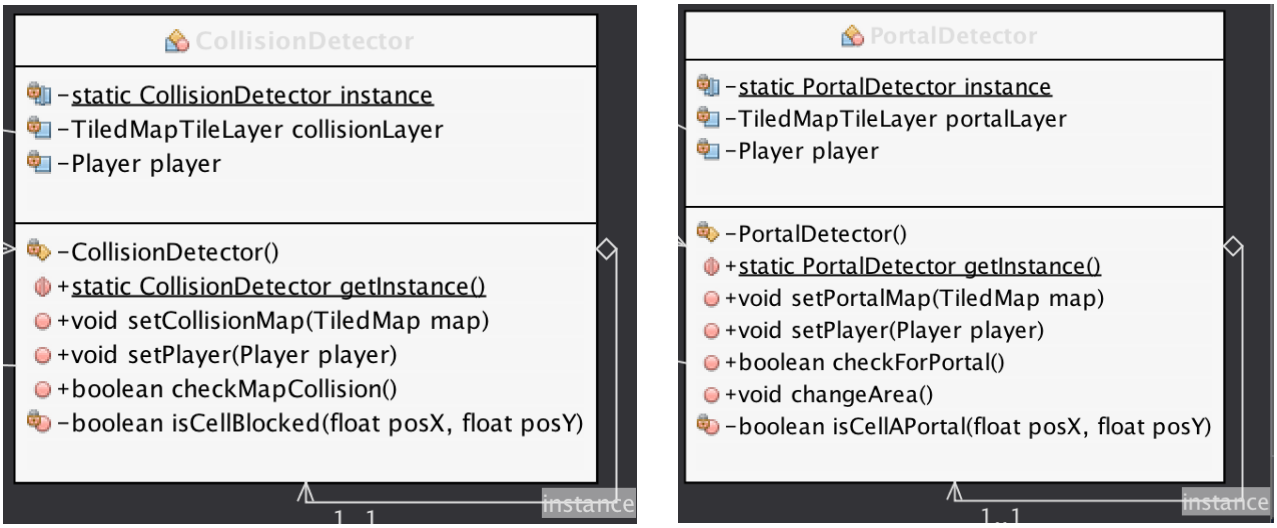

Figure 21
CollisionDetector and PortalDetector classes

Any special event in our game is represented by dedicated Layer in our TiledMap – this solution makes developing new prototype features quick and comes down to adding a proper TiledMap layers supported by corresponding detector.

```java
/**
 * Updates position of a set of points (nodes) the Player.
 * Look below for the scheme of nodes:
 *
 *          8 - 1 - 2
 *          |       |
 *          7 Hero  3
 *          |       |
 *          6 - 5 - 4
 *
 * Postconditions:
 * - the previous coordinates of nodes will be lost
 */
public final void updatePlayerNodes() {
    float[] node1 = {posX             , posY + height / 2};
    float[] node2 = {posX + width / 2, posY + height / 2};
    float[] node3 = {posX + width / 2, posY};
    float[] node4 = {posX + width / 2, posY - height / 2};
    float[] node5 = {posX             , posY - height / 2};
    float[] node6 = {posX - width / 2, posY - height / 2};
    float[] node7 = {posX - width / 2, posY};
    float[] node8 = {posX - width / 2, posY + height / 2};

    nodes[0] = node1;
    nodes[1] = node2;
    nodes[2] = node3;
    nodes[3] = node4;
    nodes[4] = node5;
    nodes[5] = node6;
    nodes[6] = node7;
    nodes[7] = node8;
}
```

Figure 22
Method updating nodes coordinates

The bare bone feature of detector system is the nodes system, implemented into the Player class. Nodes are essentially points around the player, that detect different types of tiles the player is moving through. For example, if one of the nodes detect, that the following tile is marked as “Blocked”, player will not be able to move through that tile – which is how our collision detection works. Similarly, having all the nodes detect “Portal”, player will be transferred into another area. The nodes system itself is strongly tied with our current movement system – therefore we would have to mutually adjust both if we decided to develop these functionalities further on.

ThreeDSystem

The flag functionality of our game is changing dimensions. **ThreeDSystem** is a really simple class that stores the boolean value representing the state of the system, and provides methods to change that state. Having this system turned on, the game allows for rendering additional tile layer from designated TiledMap, thus visually presenting walls on top of the 2D TiledMap. We also switch the texture of a player between its 2D and 3D versions. Thanks to the Resource loading feature, changing between those textures is fast and looks smooth.

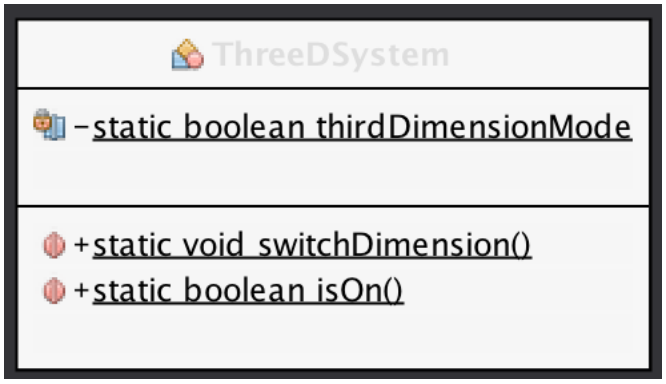

Figure 23
ThreeDSystem class

Further implementation

From this point on, the further development of this prototype is promising, but even more so is spending time re-developing the prototype to use Scene2D game library (which is also part of libGDX environment). We believe that adjusting our current code to support Screne2D would bring us even more design space and opportunities to implement many interesting features, such as AI characters, animations and lightning system. It would also potentially simplify some problems we might face, if we decided to stick to what we have got and code our ideas on the basis of it. We estimate that we would require at least 12 more months to deliver a product according to our vision and given our current work pace.

Discussion

We believe that the presented methodology is suitable for acquiring almost any kind of practical skill set, in a sense of acquisition of know-how an experience in the selected field.

The nature of its repetitive approach presents almost limitless opportunities for continual learning, given the appropriate amount of time. Each iteration forces revision of the previously acquired knowledge, which also deepen the familiarity with the subject. The slight downside to the approach might be noticed in the usage of self-inducted knowledge leading to lack of theoretical background behind experience. However, experience can be often quickly and effectively complemented with appropriate theory.

Grabbing portions of data to analyse and interpret allows for rapid researching of potentially wide field by comparing different methods that lead to the same goal. It is then possible to devote to one which seem more suitable for certain project goals. To express the idea more clearly, taking as an example ours game development research, we distinguished three technologies of developing a game application, analysed them in search for ways to create a simple game and eventually chose one to stick to.

Playing games

As a primary method in State of the Art section we used hands-on playing games. We believe that this approach has been extremely beneficial to us in terms of understanding how established video games work in general. There is no fundamental theory behind video games development, like there is in Physics. Therefore, by conducting an empirical research on existing products having industry solutions already implemented, we have extracted a know-how which we could use in our project.

Playing games as a way of learning have also highly increased our efficiency in possessing knowledge. Using unconventional methods like this one was exhilarating and joyful. Thought of playing games has been a pleasing one, and in conjunction with extracting valuable knowledge - that just could not have gone wrong.

Conclusion

During last three months, our team spend a vast amount of time researching through various ways and tools of creating games. On the basis of this research, we have created fully functional demo game using libGDX framework. Video game development requires taking into account the time-frame within which the project is created and the initial level of expertise with desired technologies the game is supposed to be made with. These dependencies in conjunction with established project goals can determine which technologies can and cannot be implemented into the project. Our experimental approach allowed us not only to distinguish most appropriate technology for our project, but also to learn the know-how of game development and to increase our software development skills in general. We also managed to create our own graphics for the game utilizing Tiled and Gimp software.

Our idea for the game have been confronted with other titles representing similar genre – during hand-on analysis on other games we could get an insight into the advanced techniques for presenting depth. Thanks to that we could come up with our own design, which represents three-dimensional scenes with two-dimensional assets such as an orthogonal camera and light and shadows concept.

References

1. http://www.polygon.com/2015/11/13/9728874/the-game-awards-2015-nominees
2. http://www.gamespot.com/articles/fallout-4-interview-we-re-probably-doing-too-much/1100-6428336/
3. http://thegameawards.com/nominees/
4. M. Saunders, P. Lewis, A. Thornhill - “Research Methods for Business Students” – “The research ‘Onion’ diagram”
5. https://en.wikipedia.org/wiki/Indie_game
6. https://lindsworthdeer.wordpress.com/2016/06/14/board-games-kickstarter/
7. https://www.quora.com/Why-is-there-such-a-big-leap-in-graphics-between-an-indie-game-vs-a-AAA-game
8. http://www.digitaltrends.com/gaming/best-indie-games/f
9. http://gamedev.stackexchange.com/questions/631/what-to-consider-when-deciding-on-2d-vs-3d-for-a-game
10. http://www.popsci.com/minecraft-sells-over-100-million-copies
11. https://en.wikipedia.org/wiki/Hyper_Light_Drifter
12. https://en.wikipedia.org/wiki/2.5D
13. https://en.wikipedia.org/wiki/Titan_Souls
14. https://en.wikipedia.org/wiki/Fez_(video_game)
15. http://www.mapeditor.org/ - Tiled software
16. http://www.reuters.com/article/us-media-internet-life-idUSTRE55G4XA20090617 - (Internet most popular information source: poll)
17. http://time.com/money/4328180/most-valuable-career-skills/ - (21 most valuable career skills)
18. https://en.wikipedia.org/wiki/Swing_(Java) - Swing
19. https://youtu.be/QM1iUe6IofM?t=869 - Object-Oriented Programming is Bad
20. https://www.lwjgl.org – LWJGL library
21. “3D game development using LJWGL” Online book by lwjglgamedev GitBook user
22. http://www.reuters.com/article/us-media-internet-life-idUSTRE55G4XA20090617 (Internet most popular information source: poll)
23. https://github.com/libgdx/libgdx/wiki/Introduction - LibGDX
24. https://github.com/libgdx/libgdx/wiki/Project-Setup-Gradle - LibGDX
25. “Java Design Patterns – A Programmer’s Approach” Pankaj Kumar @ journaldev.com
26. http://www.pixnbgames.com/blog/libgdx/how-to-manage-screens-in-libgdx/
27. dimey_javadoc