Fróðskaparsetur Føroya

5020.18 Forritanar verkfrøði (2019)

Game Engine

LUDUM: Ad Infinitum

Author
Helgi Poulsen

Supervisor
Ólavur Ellefsen

Supervisor
Julius Biskopstø

Group members
Lív Olsen
Fróði H. Hansen
Bergur I. Johansen
Herborg Kristoffersen

12th March 2019

Contents

Abstract

The difference between software engineers and programmers lies in the responsibilities and the approach to the job. Software engineers use well defined scientific principles and procedures to deliver an efficient and reliable software product while programmers tend to work alone.

This report will go into detail about the software process that incorporates five framework activities that are applicable to our game engine and to all software projects.

Foreword

In the course 5020.18 Forritanar verkfræði (2019) we are required to work as software engineers and make a game engine in *Reactjs* and meet the demands of our customer. Then write a individual report about it.

This is to prepare us for the bachelor project and give us a great understanding about software engineering.

In this report I included everything from starting knowing nothing to making a fully working game engine with two $side - scroller$[1] games by using the five framework activities in Software Engineering.

Being a part of this project helped me to understand and enhance my knowledge in what it takes to start and finish a software engineering project. Through this project I come to know about the importance of working in a functional team where everyone gave full devotion towards the work.

Acknowledgement

In any project it is very important to have guidance. In this project we had *Mr.Ólavur Ellefsen* and *Mr.Julius Biskopstø* as our supervisors. We are very grateful for the help he provided us through this project from start to finish.

[1] A side scroller is a type of video game where a side-view camera angle is used for action viewing. Side scrollers are generally in 2-D with game characters that move from the left to the right side of a screen

Chapter 1

Introduction

- Game Engine: Ludum Game Engine source code DEMO
- Games:
  - Flappy Flappy source code PLAY
  - Dino Dino source code PLAY
  - Tetris tetris source code PLAY

The *TetrisChallenge*B was a five hour competition hosted by our supervisor *Ólavur Ellefsen* on the last lecture day that we won and very proud of winning.

The main focus of this report will be about software engineering and the software process that incorporates five framework activities.

- Communication
- Planning
- Modeling
- Construction
- Deployment

I will explain about how we, as a software engineering team, went through these five framework activities to create a game engine with other requirements from our supervisors.

UML[1] is also a major part of software engineering, so in each of the five framework activities I will show some sort of UML diagram and explain about it and why it was used.

I will go into details about problems and solutions and not at least what I, working in a software engineering team, learned from this process.

Even though the purpose of this project was not about the code itself, I will show an example how some components it works.

From this point on I will call our supervisors for customers.

[1] Unified Modeling Language

Chapter 2

Problem description

As students in Science in Software Engineering without work experience, it is of great important to learn about software engineering and what it takes to become a software engineer.

Without real life experience, most students don't know what it takes to work in a team and create and deliver a software to a customer in a time frame.

With this course we get to get a simulated experience of such an event, where we have to create a product with a deadline and explain the process from start to finish.

Chapter 3

Agile

> "It does not matter how slowly you go as long as you do not stop"
>
> *Confucius*

To make our game engine a reality, we decided to go with the agile process. This was decided since the agile principles fitted us very good.

Principles behind the Agile Manifesto [1]

I *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

The main objective for us was to satisfy our customer with our game engine.

II *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

Since our time frame was quite short and everything was new for us, this was a problem for us. An example of this was when our customer wanted us to make a user interface for the game engine close to the deadline, when we had not even made the game engine work properly. We discussed with our customer, that making a UI was a project in itself, so we and customer got to the same conclusion that the requirement was unrealistic.

This doesn't mean that we where against changing requirements, even late in the process. As long as they where realistic, we had no problem.

[1] The Manifesto for Agile Software Development

Unrealistic requirements just put extra stress on the team when deadline is right around the corner

III *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

This was actually something we embraced. Our customer wanted demos once a week and this motivated us to work hard to deliver a better software.

IV *Business people and developers must work together daily throughout the project.*

This was also something we worked to achieve. We where in personal contact with our customers twice a week and we tried our best to get advice and feedback about what they thought about the software and process.

V *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*

We got all the support and trust from our customers, that we would manage to get the job done.

VI *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

We followed this by heart. We decided from the beginning that we worked together on Monday, Thursday and some Saturdays, while we meet our customers on Tuesday and Friday.

VII *Working software is the primary measure of progress.*

Our goals where always to deliver a working software for each demo and proudly showing off new features.

VIII *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*

IX *Continuous attention to technical excellence and good design enhances agility.*

By writing good code, being well organized with documentation and continuous communication with our customer and not least having a good understanding of our design I feel we had great agility.

X *Simplicity–the art of maximizing the amount of work not done–is essential.*

Our main goal was to make the game engine as simple as possible. The understanding behind this, was to make it as easy as possible to change, add or remove features.

XI *The best architectures, requirements, and designs emerge from self-organizing teams.*

We worked as a self-organizing team, everyone worked great together and put great effort in the work. One could argue that we could use a leader to report to and keep track of everything, so that all guidelines where kept all the time, but I feel we managed everything very well as a equal unit.

XII *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

With a leader in our team we could have followed this principle better, but as mention above we worked great together and where very effective.

3.1 Agile software development frameworks

we didn't follow any specific agile software development framework, but since some some of us had experience with *Scrum* and we learned about in the course *ProjectManagement*, we used it as inspiration.

- Development team

  We worked as a self-organizing team.

- Scrum master

  We where actually lacking a scrum master/leader, which I feel was a shame. I feel if we appointed a scrum master our work would be even more organized and have even better quality. Our problem was that no one wanted to take that role upon him/herself. So for future reference I would think it would be a great idea for our customers/product owners to ask the engineering teams to appoint a scrum master/organizing leader when teams get formed.

- Sprints

  We worked in iterations. Each sprint lasted 1 week where we had a demo to show our customers. Here e.g. a scrum master would be perfect to save each iteration for reflection on progress and changes.

- Sprint planning

  Again here without a Scrum master, sprint planning was lacking. We all worked very close together, so it wasn't a real issue.

  We talked between us everyday about what needed to be done and what had high priority for the sprint. I just feel it would be more professional with a real sprint planning.

- Daily Scrum

  We talked to each other every work day about what we had done the day before and what we are going to work with that day. This was very important for me that we followed. This made sure that no one would feel left out of the group and could get help if needed. We always made sure that if someone was done with his/her assignment and didn't know what to do next, the person would get a new assignment.

- Sprint reviews and retrospective

  We did not hold these two events, or not at least with the scrum principle.

  We did reflect on past sprints, but not in the Scrum manner.

Chapter 4

The five framework activities

4.1 Communication

Project requirements are collected in this activity. This framework activity is the main focus of the project managers and stakeholders. This framework activity includes communication and coordination with the clients.[1]

4.2 Planning

This framework action incorporates data about the technical work to be planned, risks to be faced, resources needed for task completion, the decision of the milestone deadline to release the product in production.[1]

4.3 Modeling

This framework phase includes the creation of product models by analyzing requirements and designing flowcharts and use cases which make developers and customers get a better understanding of the software functionalities that will achieve all requirements.[1]

4.4 Construction

In this framework activity, the actual development of the product begins with code construction and then testing to fix errors and issues.[1]

[1] Five Framework activities in software engineering

4.5 Deployment

In this framework phase, product prototypes are delivered or the completed software is delivered to the users. Feedback is obtained for getting delivered software's evaluation to add new features or future needs of the software.[1]

Chapter 5

Communication

> "The most important thing in communication is hearing what isn't said"
>
> *Peter Drucker*

5.1 First meeting with our customers

We where all very exited to meet our customers for the first time, since we knew little about the product we had to deliver. After a few hours talking to our customers, we had a requirement specification that we all could agree on.

5.2 requirements specification

This where our requirements we agreed upon:

Engineering team: **Ludum**
Customers: **Ólavur Ellefsen & Julius Biskopstø**

Start date: **15th January 2019**
Deadline: **12th March 2019**

High priority requirements

1. Create a functional game engine in React
2. Create Flappy Bird by using the game engine
3. Code coverage must be more than 80%

Medium priority requirements

1. Create a second game from the game engine of our choosing

Low priority requirements

1. Show what a possible user interface for the game engine could look like
2. Create a user interface for the game engine

Non-functional requirements

- The engine and games shall be fully functional before the deadline
- Deliver the full source code of the engine and games
- The games should be running locally and on the internet

5.3 Use Case

Figure 5.1: Use case - Game engine.

After the requirements specification, we wanted full clarity with our customers about or product, so we created a use case diagram for the customer that shows the relationship between the player/developer and the different use cases in which the users are involved.

In our use case diagram we can see we have two actors. A player and a Developer.

5.3.1 Player

The use cases for the player are what he is able to do when he runs the game. Like starting a game, jump/play it etc.

5.3.2 Developer

Here we see how the developer can use the engine to create a game. He can e.g. use entity to create a object, use a resource manager and so on.

5.3.3 Approval of the use case diagram

After finishing the use case diagram we showed it to our customers. The customers liked it, so we moved forward.

5.4 Tools

As a engineering team we decided to use these tools for the development:

- *Code editor*

  Visual Studio Code[1]

  - **Prettier**
  - **Git blame**

  For front-end development *VScode* is highly recommended. The extension prettier is very important to use, so that everyone uses the right format document. A funny example of this was when a group member didn't use the same rules as the rest of the group, so that when he did a merge request, it looked like he made changes to everything in the files he had worked on.

  Gitblame is also very good to know who to ask if not understanding the code. *Gitblame* tells who wrote the code. Not everyone used that extension.

- *Testing*

  Jest [2]

  Our customers recommended that we used *Jest* to make tests, so that is what we went with.

[1] Vscode
[2] Jest

- *Web based version control*

  GitHub [3]

  - **CircleCI**

  GitHub is easy to use and the most popular version control. *CircleCI* is very versatile, it can be used to set rules like prettier to block merge requests if the document is not correctly formatted. It also prevents merging requests that break the software. We had a small taste of this, when *circleCI* was down and bad code was merged. It took a long time to figure what was wrong, since not all members where informed that *circleCI* was not turned on. This made some of us think it that it was something wrong with the computer instead of the code.

- *Diagrams*

  - **Draw.io** [4]
  - **websequencediagrams.com** [5]

  We used *Draw.io* create all most if not all diagrams. *Draw.io* is a free online diagram software.

  Websequencediagrams was used to create the sequence diagrams.

- *Interface design application*

  Figma [6]

  We used *Figma* to design a possible *UI* to our game engine.

- *Project management*

  Trello [7]

  Trello is easy to set up and easy to use so it was perfect for us as project management.

- *Communication*

  Slack [8]

[3] GitHub
[4] Draw.io
[5] websequencediagrams
[6] Figma
[7] Trello
[8] Slack

Slack is extremely good to use as a tool for communication. It has a beautiful interface and very easy to make groups about specific topics.

...And sadly yes, we also used *Facebook*, specially in the beginning since some members had not worked with it before and did not know how powerful as a tool it is for communication in an work environment.

We had strict rules that diagrams, pull request questions related to the project and everything that was important for the project was only placed in *Slack*.

Facebook was used to ask what time we should meetup and telling if someone could not come to work that day.

Chapter 6

Planning

"By failing to prepare, you are preparing to fail"

Benjamin Franklin

6.1 Risk analysis

Risks

1. Delivering the product before deadline
2. Customer not liking the design
3. Engineering team not working as a unit
4. Unrealistic requirements added

Prevent

1. Everyone understands involved understand the requirements
2. Good communication with customer and showing demos
3. Team building activities
4. Good communication with customer

Solution

1. Regular meetings about where we are and what needs to be done
2. Showing of demos once a week
3. Openly talk about problems. Worst case remove team member
4. Good communication with customer about the new requirements

By doing the risk analysis we where always prepared of what could happen in the worst case scenario.

6.2 Object Diagram

Together with the risk analysis, we did a object model to figure out how we wanted our engine to look like.

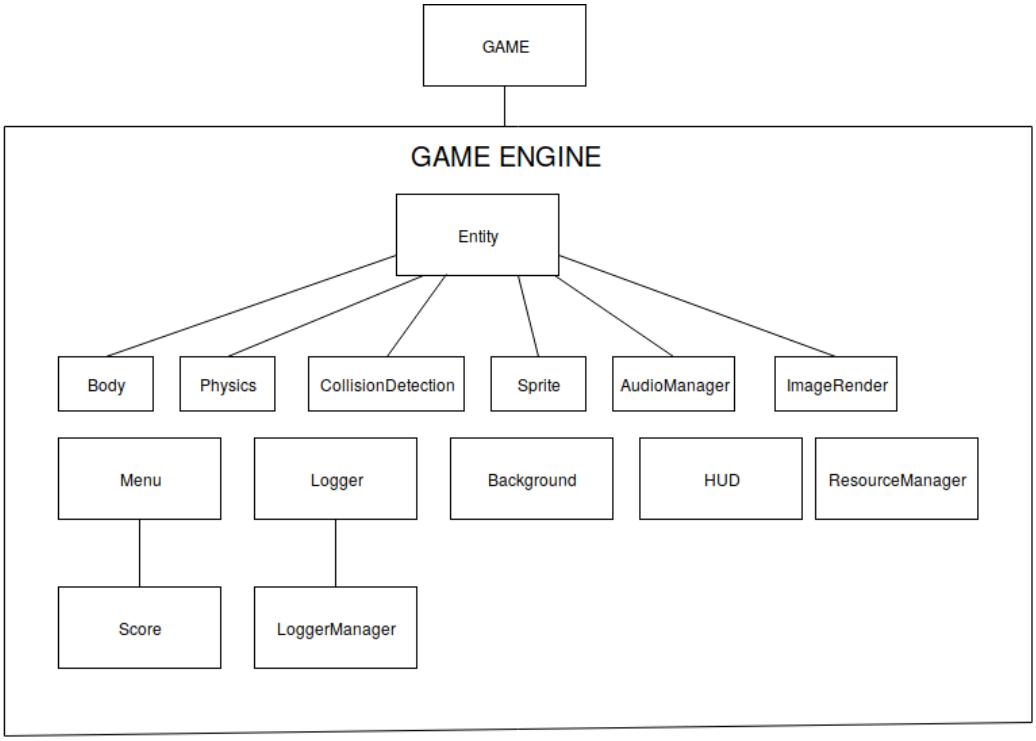


Figure 6.1: Object Diagram - Game engine.

From this object diagram6.1, we could get a clearer picture of how we wanted our engine to look like. This also prepared us for making assignments for our backlog and even give them priorities.

This diagram6.1 is also good to set milestone. e.g. first milestone is to create a *Entity* that had a *Body*, *Physics*, *CollisionDetection*, *Sprite*, *AudioManager* and *ImageRender*.

The next milestone contains *Background* and *ResourceManager*. Then *Menu*, *ScoreBoard* etc.

> "What one programmer can do in one month, two programmers can do in two months"
>
> Fred Brooks

The diagram6.1 is also a good way to put people on different task since everyone cant work on the same thing. Example someone can start with *Entity* while another person begins working with *ResourceManager*.

6.3 Bad planning

Bad planning can have disastrous consequences.

When we started working, we had little to no knowledge about *React* and game engines. In the beginning we talked a lot to our customers about what they expected it to look like. In response to this we made lots of diagrams to show of our plan. The problem with coding is that there are endless ways to do the same task.

We started out just doing the game without a game engine, this reason was just to learn how to use the *React* library and create something.

Creating *Flappy* didn't take long, so we where quite confident on how to create the game engine and that our model was the right approach.

After a few weeks we figured out that our template of a game engine was quite bad. This came to light when our customer *JuliusBiskopst* showed a really great template of how to create a basic game engine.

We had to change lots of things and used a lot of time restructuring our engine. If we figured this out later in the process our product would be not close as good as when it was delivered.

The blame for this was lack of communication with our customers. We knew both our customers have lots of experience in creating game engines and coding in *React*, but after creating *Flappy* without a game engine in a timely manner, we got carried away and communicated to little with our customers about our approach on how the design of the engine should look like.

Chapter 7

Modeling

7.1 Game World

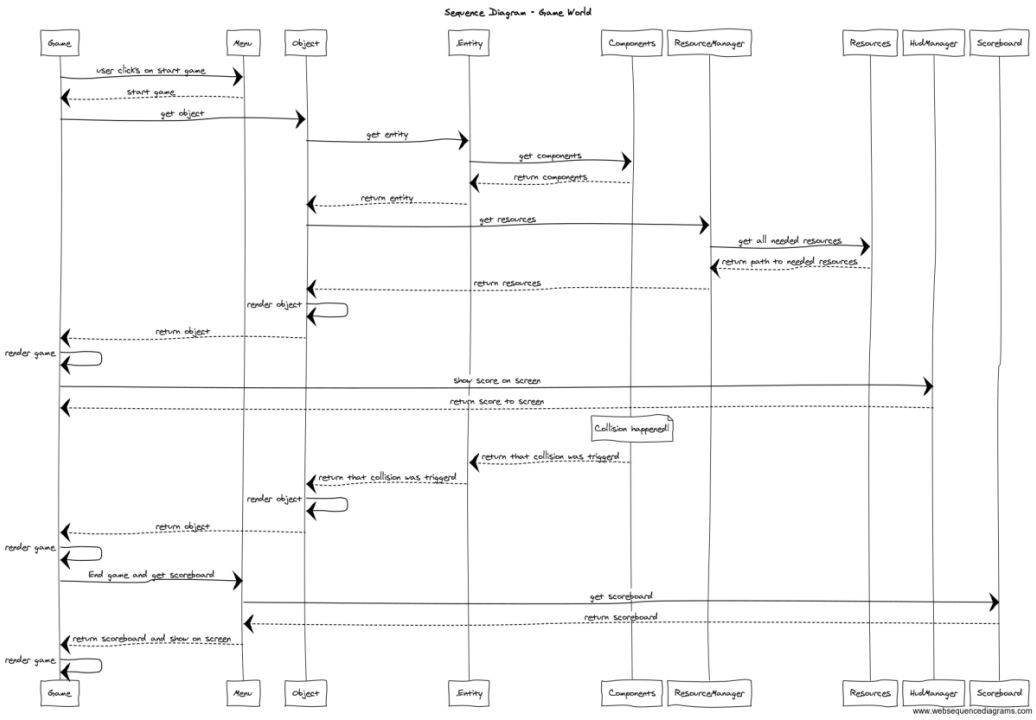


Figure 7.1: Sequence Diagram: Game World

To give our customers a better understanding of our product, we created sequence diagrams, also known as event diagrams, to give a better understanding of the game engine functionalities.

Sequence diagrams can sometimes be hard to understand. To read a sequence diagram the user shall read downwards.

In7.1 we see all the events in our *GameWorld*.

If we start with game and read downwards we see that the first event is user clicks on start game, then *Menu* returns start game. Next the game gets a object, the object gets a *Entity*. *Entity* gets components (*Physics*, *collisionDetection* etc.) those components get returned to *Entity*. Then object gets resources etc.

7.2 Render Loop

I personally prefer smaller sequence diagrams, since they are easier to read and less confusing. Like 7.2

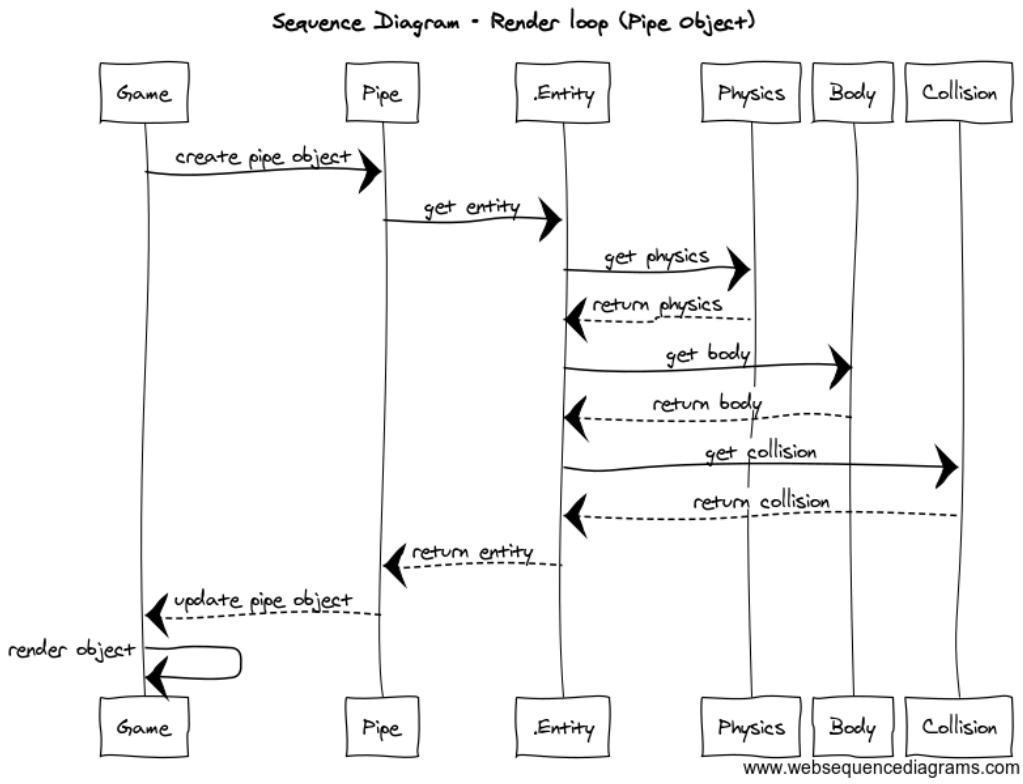


Figure 7.2: Sequence Diagram: Render loop (object)

Here7.2 we see a much clearer picture of how an object works.

Game creates a object, in this case a pipe. *Pipe* gets a *Entity*, *Entity* gets *Physics*, *Body* and *Collision*. *Physics*, *Body* and *Collision* gets returned

to *Entity*. *Entity* gets returned to *Pipe*, and *Pipe* gets updated. Then the object get rendered.

As said here its much easier see whats going on compared to a huge event diagram.

Chapter 8

Construction

"The road to success is always under construction"

Arnold Palmer

8.1 Architecture

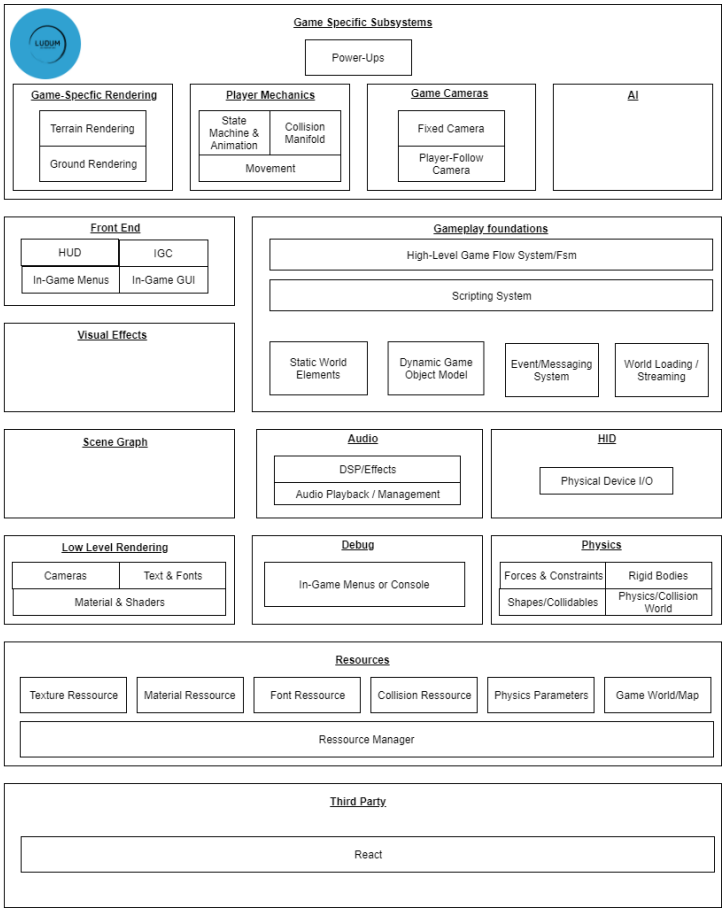


Figure 8.1: Architecture Diagram: Game Engine

This8.1 is how we imagine how the architecture of our game engine would look like so that it would be able to run $side-scrollers$ like $Flappybird$. Our version is though much simpler, that I will show with a class diagram.8.2

8.2 Class Diagram

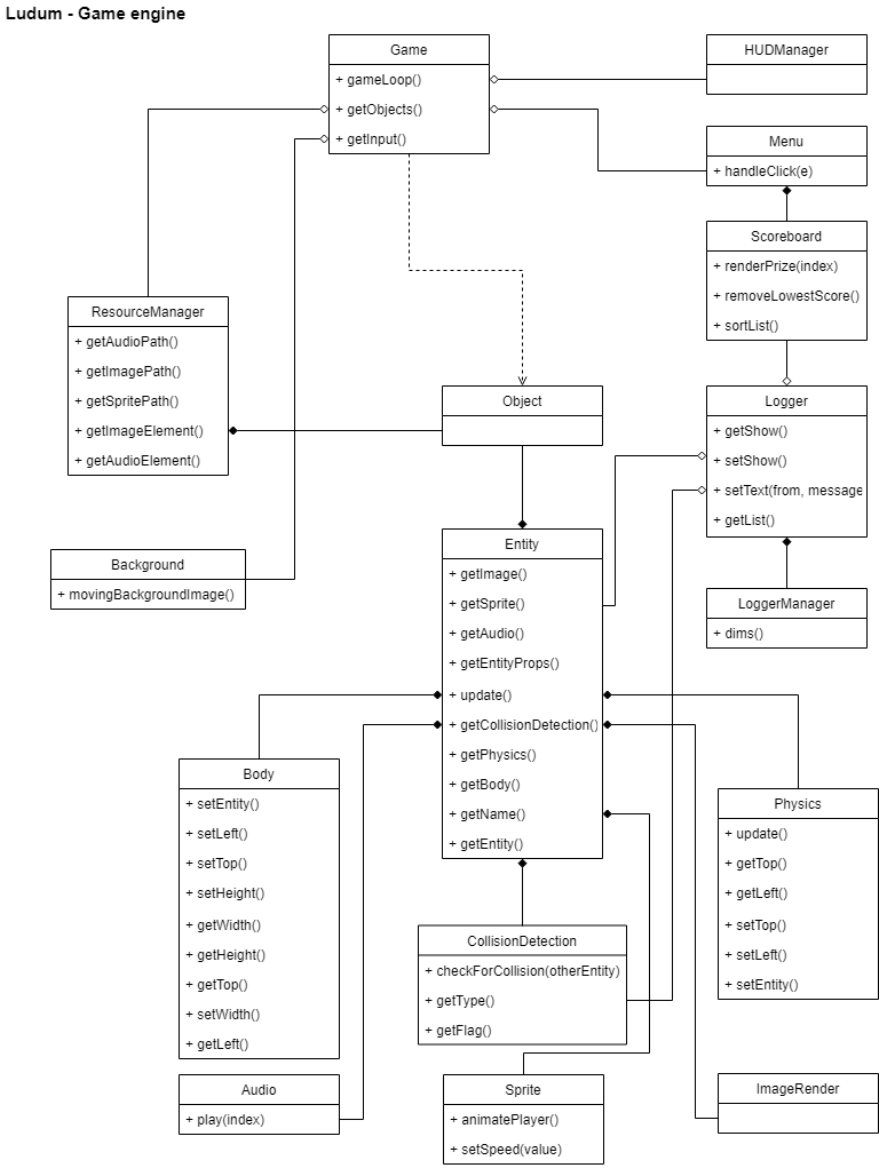


Figure 8.2: Class Diagram - Game World

This8.2 is the static structure of our system. The game needs to have three operations.

- gameLoop()
- getObjects()

- getInput()

From there the developer can use what ever classes he like as he wishes.

This class diagram8.2 is very important for the developer of the game, since it shows a much better overview of what the possibilities are with the game engine.

We can for example see that in class *CollisionDetection*. Operation $checkForCollsion(entity)$, that when we are checking for collision by using another entity as argument. This means that $entity1$ compares its properties with $entity2$ properties. Then returns a *boolean*.

We have given all operations proper names so that the user understands exactly what the operation does. A small other example is in *Menu*, operation $handleclick(e)$. Without looking at the code you can guess that depending on the user click, different tasks get performed.

8.3 Tests

Since this project is not about coding, I wont get into details about how to do the tests. Instead I am just going to show how much code coverage we have.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	85.42	73.08	89.36	85.42	
gameEngine	100	100	100	100	
Entity.js	100	100	100	100	
gameEngine/components	80.85	50	93.1	80.85	
AudioManager.js	80	50	100	80	8
Body.js	100	100	100	100	
CollisionDetection.js	100	100	100	100	
ImageRender.js	40	100	50	40	10,19,26
Physics.js	100	100	100	100	
Sprite.js	66.67	0	75	66.67	... 66,67,69,72,75
utils	57.14	100	57.14	57.14	
Logger.js	25	100	25	25	6,10,18
ResourceManager.js	100	100	100	100	

Test Suites: 7 passed, 7 total
Tests: 31 passed, 31 total
Snapshots: 0 total
Time: 3.28s
Ran all test suites.

Figure 8.3: Code coverage

We are very pleased with our tests and code coverage. We showed one of our customers in the *Tetrischallange* our code coverage and he was pleased.

Chapter 9

Deployment

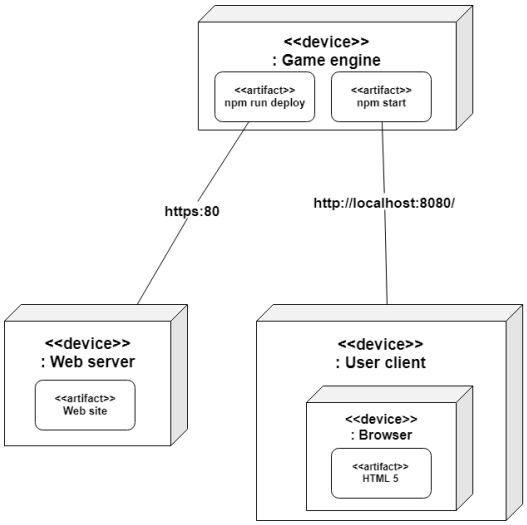


Figure 9.1: Deployment Diagram

To get the final feedback and evaluation of our product, we delivered it to our customers via public *GitHub* repositories. This means that the customer can download the our product and run it locally.

We also provided the customer with web links for live demonstration of the game engine and all the games.

To show our deployment without using words, we created a simple deployment diagram for our customers on how they will receive the product.9.1

A how to for the game engine is also provided. It can be found in *LudumGameEngine* in the folder extra. It can also been seen in *README* in *LudumGameEngine* repository.

9.1 Future needs of the software

We created a very simple design of what a user interface for our game engine could look like.

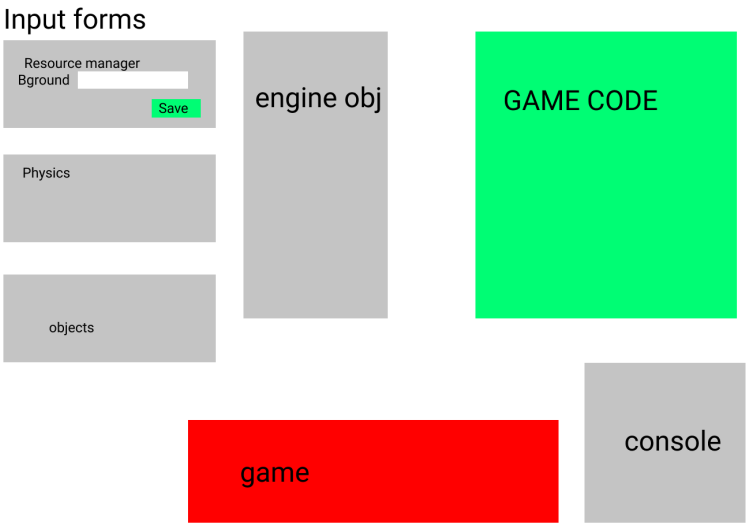


Figure 9.2: Graphical UI

It doesn't look like much9.2, but it gives an taste of what a user interface for the game engine could look like.

Since we felt this was a project in itself we didn't spend more time on it and put all our focus on our main objective. Creating the game engine and the two games.

Chapter 10

Code example of an component

10.1 Diagrams

The game engine has many of components that are used for different tasks.

Here i am going to show of how our MenuC component works with a few diagrams.

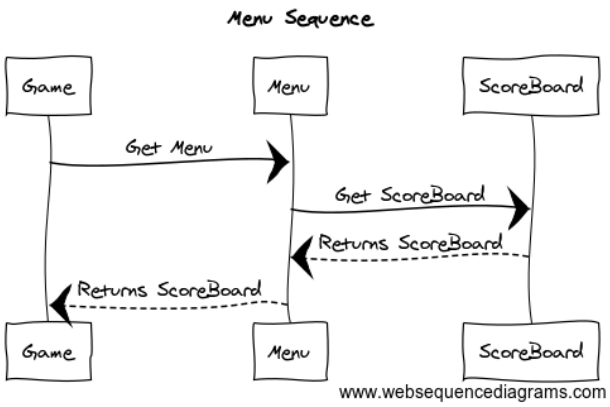


Figure 10.1: Sequence - Menu

Here10.1 we can see that *Game* gets *Menu*, *Menu* gets *ScoreBoard*,*ScoreBoard* gets returned to *Menu* and then *Menu* gets returned to *Game*.

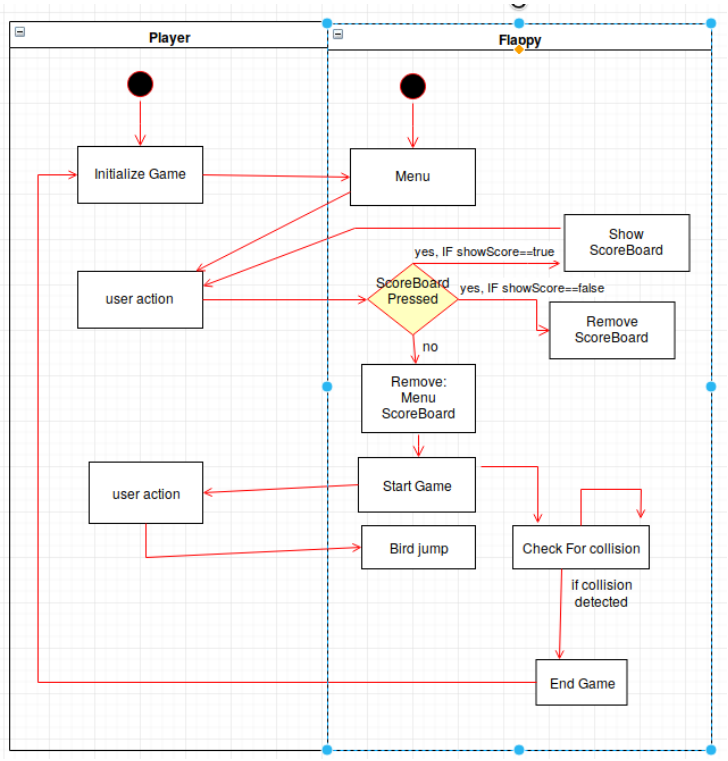


Figure 10.2: Activity diagram - Flappy using Menu component

In this diagram10.2 we can see how the menu is used via user actions.

The menu gets shown when *Flappy* gets initialized, then if user presses *ScoreBoard*4, *ScoreBoard* component gets shown. If the button is pressed again *ScoreBoard* gets removed again.

If user action is anything else, the *Menu* and *ScoreBoard* get removed.

10.2 CodeC

React, *menu.css* and a *scoreboard* components get imported.

10.3 Constructor

In the constructor this.states are set:

```
1     this.state = {
2       menuItems: [{ text: "Score" }],
3       showMenu: true,
4       showScore: false
5     };
```

in *menuItems* the user can set as many as he wants. In this case its just score. *showMenu* is set to *true* and *showScore* set to *false*.

Then handleClick gets binded like this:

```
    this.handleClick = this.handleClick.bind(this);
  }
```

10.4 Variables

10.4.1 let scoreBoard

This variable returns *ScoreBoard* component

10.4.2 let gameMenu

This variable returns the a Menu-text and Menu buttons, depending on how many *menuItems* there are. *menuItems* is the *constmenuItems* in *render*()

10.5 Methods

10.5.1 function MenuItem(props)

This function returns a button where *id* is set, and a *handleclick* is set for eat item.

Then it sets the button text to uppercase.

by using the *map*() method in *render*, we can set buttons for all the elements in the *menuItems* and give them *handleClick*.

So the menu "list" works in a very dynamic way, where the user only needs to insert a new *menuItem* and then a button with *handleclick* etc. gets generated automatically with this function and:

```
  render() {
    const menuItems = this.state.menuItems.map((item, index) => (
      <MenuItem key={index} item={item} handleClick={this.
    handleClick} />
    ));
```

10.5.2 handleClick(e)

handleClick(*e*) is the event handler for the buttons. *e* is the props.item.text of the button, so in this case, *if e* === "*Score*", set the *state* of *showScore* to the opposite.

10.6 return();

Here is what gets returned from *Menu* Component depending what the *props* and *states* are of *showMenu* and *showScore*.

```
1         {this.props.showMenu ? gameMenu : this.state.showMenu}
```

Here menu gets shown or not depending of the *props* of the *boolean* that gets passed down from the *Game*.

```
1         {this.state.showScore && this.props.showMenu
2           ? scoreBoard
3           : this.props.showScore}
```

here we check the *state* of *showScore* and the *props* of *showMenu*. If both are *true*, show the *scoreBoard*.

Chapter 11

Conclusion

By doing the software engineering process, we delivered a fully working game engine that can create side-scrolling games. We also delivered two fully working side-scrolling games:

- Flappy
- Dino

We also won the *TetrisChallange* by using everything we have learned and using our game engine.

We failed to do the user interface and present a beautiful design of what it could look like by using *Figma*, but I feel everyone in our engineering team is still very pleased with our results and what we have achieved.

We have yet to get feedback from our customers, but they have liked what we have shown so far, so hopefully they will be happy with our final product.

Appendices

Appendix A

Deployment

- Game Engine: Ludum Game Engine source code DEMO
- Games:
  - Flappy Flappy source code PLAY
  - Dino Dino source code PLAY
  - Tetris tetris source code PLAY

Appendix B

Tetris Challange

Avbjóðing

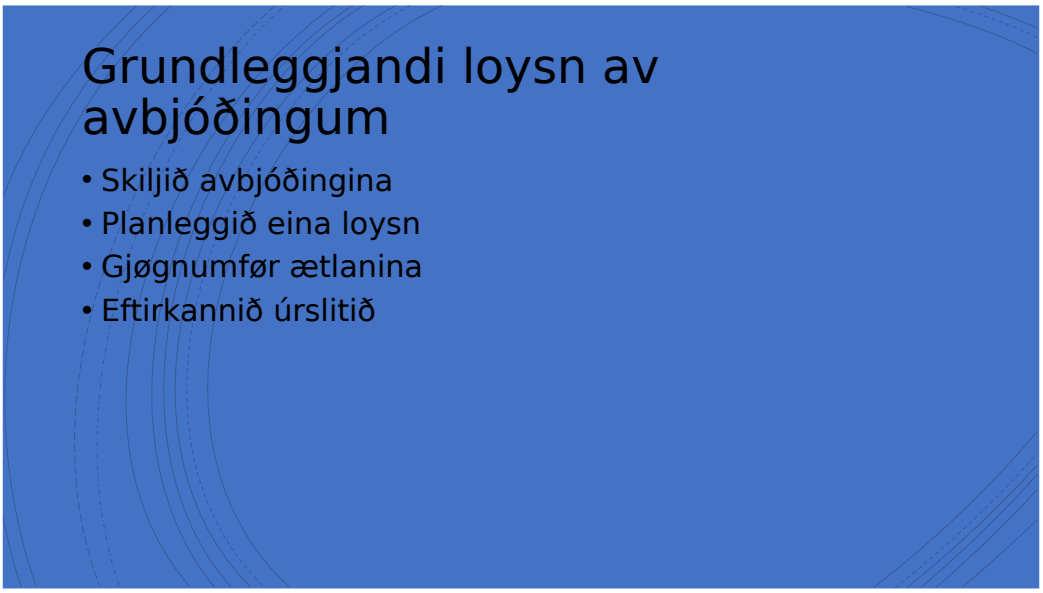
Forritaverkfrøði

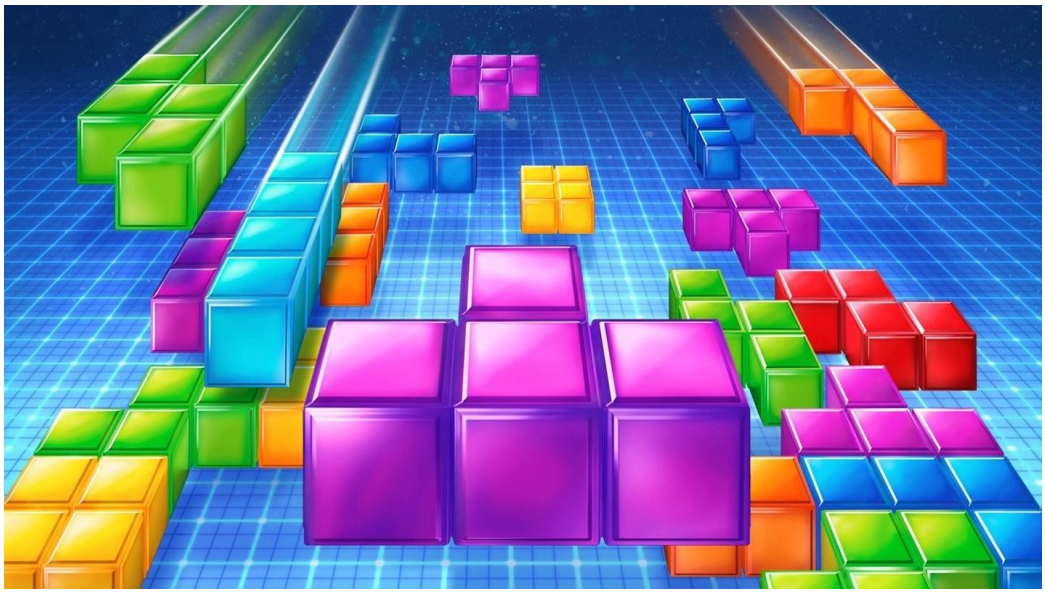
Ólavur Ellefsen – 8. mars 2019

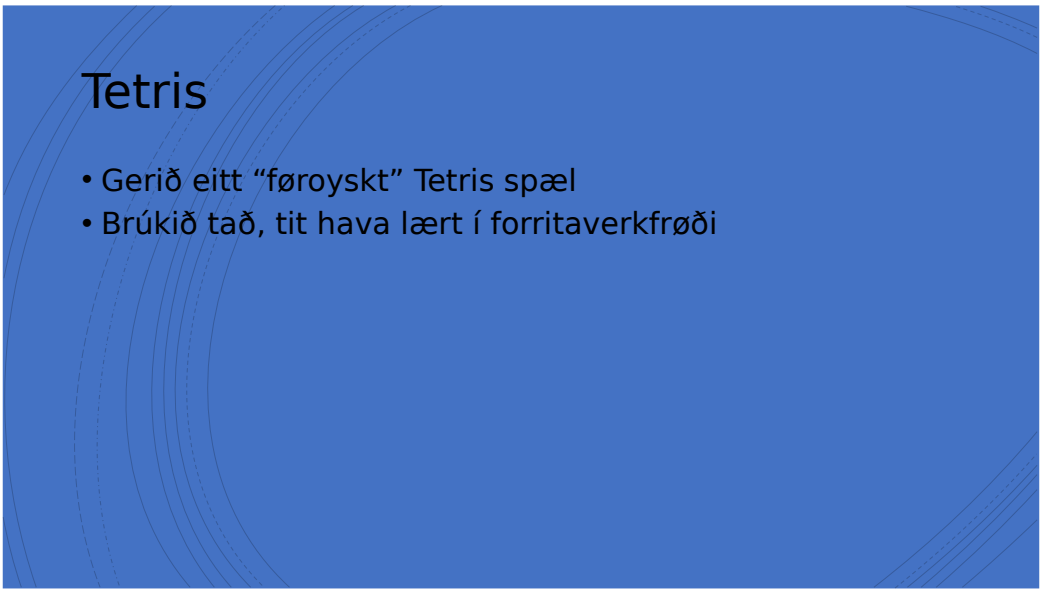
FRÓÐSKAPARSETUR
FØROYA

Yvirskipaðar forritatilgongdir

- Samskifti (kundasamstarv og innsavnan av krøvum)
- Planlegging (gera arbeiðsætlan, lýs tøkniligar váðar, lista neyðugt tilfeingi, lista hvørji produktir skulu framleiðast og ger tíðarætlan)
- Modellering (skapa modellir at hjálpa mennarum og kundum at skilja krøv og formgeving av forritum)
- Konstruktión (koding og testing)
- Veiting (ritbúnaður veittur til kundaeftirmeting og –afturmelding)


Grundleggjandi loysn av avbjóðingum
• Skiljið avbjóðingina
• Planleggið eina loysn
• Gjøgnumfør ætlanina
• Eftirkannið úrslitið




Tetris
• Gerið eitt “føroyskt” Tetris spæl
• Brúkið tað, tit hava lært í forritaverkfrøði

Evstamørk

10:00 Kravfesting: Use case diagram og tekstur. Non-funktionel krøv.

11:00 Ætlan fyri arbeiðið: User stories. Tíðarætlan. Váðar. Veitingar.

12:00 Formgeving: UML diagrammir. Grafisk útsjónd.

13:00 Leinki til keldukodu

14:00 Kodan fryst


Kapping – max 10 stig fyri hvørt punkt
1. Kravfesting kl 10:00
2. Ætlan fyri arbeiðið kl 11:00
3. Formgeving (Figma ella líknandi) kl 12:00
4. UML modellir av forritinum kl 12:00
5. Keldukodugoymslan tøk kl 13:00
6. Skjalfest keldukoda kl 14:00
7. Forritið koyrir hjá kundanum kl 14:00
8. Unit- og integratiónstestir kl 14:00
9. Loysnin uppfyllir endamálið kl 14:00
10. Tilgongdin stýrd væl (váðastýring) kl 14:00

Appendix C

Code

C.1 Menu.js

```
import React from "react";
import "../../style/menu.css";
import ScoreBoard from "./ScoreBoard";

class Menu extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      menuItems: [{ text: "Score" }],
      showMenu: true,
      showScore: false
    };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick(e) {
    if (e === "Score")
      this.setState({
        showScore: !this.state.showScore
      });
  }
  render() {
    const menuItems = this.state.menuItems.map((item, index) => (
      <MenuItem key={index} item={item} handleClick={this.
    handleClick} />
    ));

    let scoreBoard = (
      <div className="wrapper-scoreBoard">
```

```
31          <ScoreBoard />
32        </div>
33      );
34      let gameMenu = (
35        <div className="wrapper-menu">
36          <div className="menu-items">
37            <div className="menu-text">Press any key to start</div>
      {menuItems}
38          </div>
39        </div>
40      );
41
42      return (
43        <div className={`menu ${this.props.position}`}>
44          {this.props.showMenu ? gameMenu : this.state.showMenu}
45          {this.state.showScore && this.props.showMenu
46            ? scoreBoard
47            : this.props.showScore}
48        </div>
49      );
50    }
51  }
52
53  function MenuItem(props) {
54    return (
55      <button
56        className="menu-item"
57        id={props.item.text}
58        onClick={() => props.handleClick(props.item.text)}
59      >
60        {props.item.text.toUpperCase()}
61      </button>
62    );
63  }
64
65  export default Menu;
```