UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA
(FINAL PROJECT PAPER)

RAZVOJ SPLETNE APLIKACIJE ZA VEČ UPORABNIKOV: DEZERTIO IGRA PREŽIVETJA (DEVELOPMENT OF A MULTI-USER WEB-BASED APPLICATION: DEZERTIO SURVIVAL GAME)

IRHAD ELEZOVIQ

2020
ZAKLJUČNA NALOGA (FINAL PROJECT PAPER)
ELEZOVIQ

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Razvoj spletne aplikacije za več uporabnikov: Dezertio igra preživetja

(Development of a multi-user web-based application: Dezertio survival game)

Ime in priimek: Irhad Elezoviq
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, avgust 2020

Ključna dokumentacijska informacija

Ime in PRIIMEK: Irhad ELEZOVIQ

Naslov zaključne naloge: Razvoj spletne aplikacije za več uporabnikov: dezertio igra preživetja

Kraj: Koper

Leto: 2020

Število listov: 48 Število slik: 7

Število referenc: 10

Mentor: doc. dr. Peter Rogelj

Ključne besede: splet, življenjski cikel razvoja programske opreme, razvoj iger, igra za več igralcev

Izvleček: Nenehno izboljševanje in napredek spleta sta uporabnikom omogočila uživanje v bogati interaktivni vsebini in poenostavila postopek razvoja spletnih aplikacij. Razpoložljive tehnologije uporabljamo za izdelavo in predstavitev celotnega SDLC (življenjski cikel razvoja programske opreme) spletne igre za preživetje. Igra ponuja pristop žanra preživetja od zgoraj navzdol, kjer mora igralec zbrati dovolj sredstev, loviti in preživeti noč. K zapletenosti pripomorejo tudi drugi igralci, katerih dejanja vplivajo na potek celotne igre.

Key document information

Name and SURNAME: Irhad ELEZOVIQ

Title of the final project paper: Development Of A Multi-user Web-based Application: Dezertio Survival Game

Place: Koper

Year: 2020

Number of pages: 48 Number of Figures: 7

Number of references: 10

Mentor: assist. prof. Peter Rogelj, PhD

Keywords: game-development, web, multiplayer, SDLC

Abstract: The constant improvement of the Web and its incessant progress has allowed users to enjoy rich interactive content and has made the development of web-based applications a straight-forward process. We use the available technologies to build and showcase the entire SDLC of a web-based survival game. The game offers top-down approach to the survival genre where the player must gather enough resources as well as hunt and survive the night. Other players also add to the complexity because their actions affect the gameplay as well.

ACKNOWLEDGMENTS

I am grateful to my mentor doc. dr. Peter Rogelj for his extensive feedback and to bobbylolly for the game assets.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTML	Hyper Text Markup Language : markup language for documents displayed in a web browser
UML	Unified Modeling Language : general purpose modeling language
DFD	Data Flow Diagram
UI	User Interface

1 INTRODUCTION

The Web was developed in 1989 by Tim Berners-Lee at The European Organization for Nuclear Research (CERN). What inspired this discovery was the fact that at CERN, they kept losing information due to the high turnover rate of employees. The way information was shared was primarily by face-to-face communication and newsletters which meant that when an employee left, a lot of information was lost. Projects at CERN were dynamic and information was constantly changing - this in turn meant that ineffective information management caused duplicated effort. Tim Berners-Lee went over the existing solutions at the time and finding them inadequate decided to propose a global hypertext system as a way to manage information [1].

Initially the system was called Mesh but afterwards he decided to call it the "World Wide Web". Furthermore, he made the first draft for HTML which is the hypertext markup language of the modern Web. Since then, the Web has revolutionized the way we share information, form social connections, work, communicate and entertain ourselves. In 1993 the World Wide Web was already made available on the public domain and enormous amount of users began using it. By 1995, 18 million American homes were online. As a result of this, the World Wide Web became an important component of the Information Age. From the standpoint of tech companies, the Web clearly ushered in a new age and they put their best effort into utilizing its capabilities. Companies and users wanted to employ the benefits of the Web so they started developing websites for it. As of right now, there are 1.5 billion websites [2].

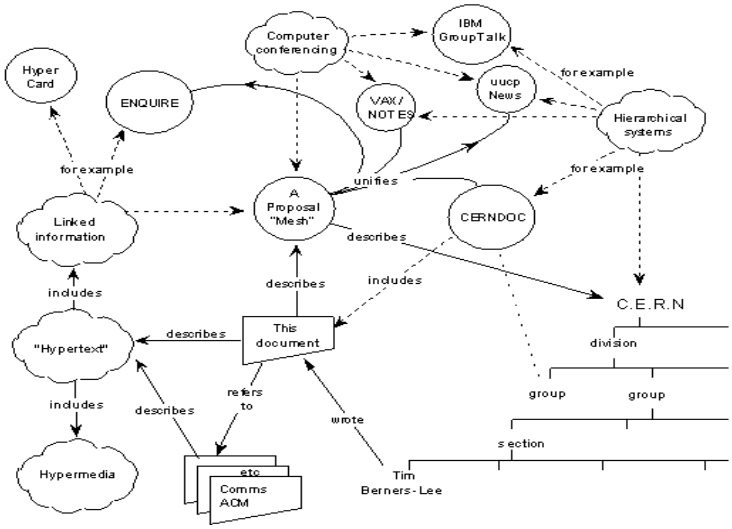

Figure 1: Tim Berners-Lee's Initial Proposal

Furthermore, the development of HTML5 allowed browsers to natively include and handle multimedia and graphical content. This, in turn, has made the Web into a platform that can offer incredibly rich user experiences and games. Players find it very appealing that they can

try out games that do not require any installation and are playable straight from the browser [2].

1.1 Motivation

Games have been part of human history for thousands of years. They have been important as cultural and social bonding events. The board game senet is older than 5000 years and is considered to be the oldest board game in existence [3]. Unsuprisingly, with the advent of the information age games have stepped into the virual world as well.

It is widely considered that the first video game created is the infamous 1958's “Tennis for Two” featuring moving graphics on an oscilloscope [4].

Figure 2: Tennis for Two on a DuMont Lab Oscilloscope Type 304-A

Since then, owing to the shrinking of computers and their ubiquity, lots of programmers began to create games leading up to the release of "Spacewar!" in 1962 which was one of the earliest digital computer games. Engineers and computer enthusiasts were becoming aware of the commercial viability of game development and thus in 1972 Ralph Baer created the first gaming console known as the "Magnavox Oddysey" [5]. Since then, game

development has become a massive industry and has generated $135 billion in profits in 2018 and is expected to reach $180 billion by 2021 [6].

The evolution of the Web made it a viable platform for development. As a consequence of increasing demand for games it began to be utilized as a gaming platform. Browser games of various genres were developed either in single-player or multiplayer mode. The benefit of browser games lies in the fact that they can be played on different devices and operating systems making them portable.

When it comes to market size the numbers look really promising. In research done by *Internet World Stats*, the number of internet users in the world is roughly 4.5 billion people [7]. Another study done by *Earnest* shows that in 2015 roughly $90 billion was spent on video games [8].

But when we look into browser games and different free online multiplayer games, the percents rise dramatically. According to a study conducted by *Statista,* roughly 90% of children in the UK report playing online video games, a great number of which include browser games [9].

We have decided on building a browser game in the survival genre because this genre has been in the interest of game developers in recent years. This is due to the fact that survival games are engaging and offer the player a wide range of possible strategies. The player interacts both with the environment and other players and they can choose to try to survive on their own and kill their competition for resources or they can team up with other players and share resources.

The amount of possible strategies the player can choose from adds a certain complexity to survival games that make them stand out among other genres of video games. The fact that the game is a multiplayer online game in a public server adds another layer of that complexity because each player needs to gauge the rest of the players, deduce what their possible strategy is and adjust their own strategy accordingly. This is why the in-game communication is vital for coordination and cooperation.

1.2 Inspiration

Dezertio is heavily inspired by other games in the survival genre like "Don't Starve Together", "Minecraft", "Terraria" etc. Multiplayer survival games offer engaging gameplay in an open hostile world where the player needs to gather resources, craft tools and cooperate

with other players in order to survive as long as possible. Additionally, they indirectly teach players to make effective decisions under time-pressure and to cooperate.

1.3 Objectives

Defining objectives and certain requirements is crucial before undertaking to build a game. It's important that the objectives are well defined and that they are completed. Thus, we define the main objectives for the browser game as follows:

- Developing a multiplayer browser game where the player's survival or demise is influenced by other players.
- Developing a client-side which renders a shared scene on each connected client
- World design – The world should be large, aesthetically pleasing containing resources and mobs which are spaced out uniformly
- Developing a server-side which tells the client when to regenerate resources and updates the shared world according to actions taken in it
- In-game balancing – The game needs to be balanced and each player needs to have a fair chance of survival and thriving

Creating a learning experience for players by indirectly teaching them that cooperation is more beneficial in the long run than competition

2 GAME DESCRIPTION

In this chapter, we will broadly describe how the game is seen from the perspective of the player and go into some details and specifics of the game such as items that can be crafted and hostile animals that the player will encounter in the game and the items that they pick up after killing them.

2.1 Overview Of Dezertio

The player immerses themselves as an explorer in a foreign world that imitates the conditions of a desert biome. It's a well-known fact that the most difficult places to survive in the world are deserts. Deserts make up a third of the world's land area and they are some of the lowest population density areas thanks to the harsh environment, scarce sources of food and water and the great of number of deadly predators.

The player spawns in a random spot in this barren world with an empty inventory. They can spawn at any point of the day-night cycle of the game which lasts 10 minute each. This means that they must decide what they need to do first and what should be their main priority, find food, water or gather resources in order to craft items. During the day, the scorching temperatures are among the biggest threats for the player as they must find a water source to battle the heat and dehydration, on the other hand the night offers another set of problems for the player. At night, the temperatures are very low and without shelter or a fire, there is a great risk of hypothermia.

Each time that the player logs in, they start from the beginning regardless if their last log in ended with their death or exiting the game. If the client exists the page before they died, their character will remain idle in the game and will eventually die from the elements, hostile animals or other players. The fact that with every log-in the player has to start from the very beginning and the fact that there are different players playing at different times will inevitably require the player to adjust their strategy to the established environment in the server by the rest of the players currently present. This will ensure that the person playing will have a distinctive and original experience every time they play the game even though the world itself hasn't changed

This first day for many new players would be the hardest as they learn what the world of the game has to offer and they're learning how to develop the best strategy for survival. After the player has managed to survive during the first couple of days and they have covered their basic necessities, they must continue to develop. They have to decide which development path to take, are they going to be surviving on their own or are they going to attempt to find

a player or players to join with and survive together. At this stage of the game, survival is easier than it is during the first couple of days but for the sake of the development the player has to put themselves in danger and gather even more resources, craft weapons and fight against the predators. Killing animals offers food but besides that, they offer other resources that can be used in different crafting recipes.

We're now going to take a look at the different components that make up the world of the game and compile a list of the main strategies that can be employed for successful survival in the game.

2.2 Building The World

The first step taken in developing the game is to build a space where all the interaction happens. The world of the game consists of several components that together build its environment. These components are hard-coded into the game.

- Oasis is the only source of water for the player and this makes it a vital part of their survival. They are scarce and they are gathering spots for a lot of players who may have bad intentions. In the prototype version of the game, the oasis doesn't dry up and is limitless.

- Cacti are spiky plants with a flower on top, containing an edible fruit in them which is also the only food source for the player in the first couple of days of their survival.

- Trees, stone ores and gold ores are the main resources for crafting that the player can use in a variety of different crafting recipes for tools, weapons and other items that would help them survive and make progress in the game.

2.3 The Player

The avatar of the player is a rectangle-figure with circles on both sides imitating hands that can move when the player hits something. The player can move through the world using arrow-keys or ASWD buttons of the keyboard as well as the following combinations for diagonal movements - arrow keys left + up, right + up, right + down, left + down or W + D, W+A, S+A, S+D.

In addition to that, left mouse click moves the character's hand which is used when mining or using weapons to hunt mobs or fight with other players. When the player is holding one of the various craftable tools such as pickaxe or sword, the radius that they can reach

increases based on the tool that they hold. The character is only able to hold one tool at a time and the player's nickname is displayed above their head which is visible to all players.

2.4 Gauges And Menus

As part of the UI of the game, the player can interact with the inventory menu, the crafting menu as well as have overview of the gauges in the game.

- Health/thirst/hunger/temperature gauge - The four are displayed at all times on the top left corner of the screen. In order to remain alive, the player has to keep all four at reasonable levels. Health is decreased when the player is injured by another player, a hostile animal or the other three gauges are past their critical point. The player will continue losing health until their death or they find a way to improve their condition.

- Inventory menu - The items that the player obtains via gathering or crafting are automatically added to their inventory which is represented by 8 squares at the bottom center of their screen. The 8 slots can hold 8 different items with multiple items of the same kind taking the same slot. The weapons and tools can be selected via mouse click or pressing the number on the keyboard representing the number slot this item is in their inventory. For example if the sword is on slot 3, by pressing 3 on the keyboard, the player will equip the sword. Unequipping the item currently held is done the same way you would equip it.

- Crafting menu - The crafting menu is displayed on the top right corner of the player's screen and it will be displaying only the items that the player is capable of crafting with the resources currently in their inventory. The crafting is done by clicking on the desired item which has a specific crafting time during which the player is not able to do anything else except move with the ASDW buttons. After the crafting is completed, the item will appear in an empty slot in their inventory unless such an item already exist in which case it will increase the number of that item by one in the players inventory. If the player already has 8 different items in their possession and they want to craft a 9th, when clicking on that item in the crafting menu, he will receive a message on the top of their screen that their inventory is full and they need to use up or delete one of their resources by right-clicking on them in their inventory.

- Chat menu – The chat menu is displayed on the bottom left corner, displaying the last few messages sent by any player in the server along with their nickname and a time-stamp. In order to send a message, the player has to press enter, type their message and then hit enter again.

2.5 Hostile Animals

The third major part of the in-game world is the hostile animals or mobs. In the game, there are 3 different types of hostile animals - hyenas, elephants, and scorpions. These hostile animals spawn in specific locations across the space of the world and will respawn after they have been killed. They are an additional source of resources that the player can use to craft different tools essential for their survival. However, due to their high value, the price for obtaining them is high as well - the player is risking their life.
Each of the three animals existing in the world is planned to have their own different behaviors which the player will discover and plan accordingly to when deciding to fight against them.

- The hyena is the more simple of the three mobs in the game, the hyena is difficult to kill and it takes the player many swings with the sword to get the job done, even when using the most powerful weapon in the game. The hyena has a lot of health and when the player gets within its radius, it would start chasing them relentlessly until they manage to escape the said radius or one of them is dead. In the first few days of their survival, the player will need to do their very best to stay away from this animal but once they've managed to get a weapon, they will most definitely want to attempt hunting one because of the valuable hide the hyena drops. The skin can then be dried and used for crafting. The movement speed of the hyena is only slightly slower than the default speed of the player but they deal a lot of damage to the character and if the player isn't careful and back away as they're hunting, they will be dead long before they can kill the mob.

- The scorpion has a smaller amount of health points than the hyena but fighting against it has its own set of challenges. The scorpion is planned to have the same movement speed that the character has, which makes it virtually impossible to escape from once you have entered its radius and it has started chasing you. However, when it stings you, it does very little damage. But even that small sting could kill the player as its venomous. From that moment on, every few seconds, the player will keep losing health until they're dead or they consume a special potion crafted with cacti fruits. Killing the scorpion would be profitable for the player because of the venom gland that it would drop upon its death which could then be used in crafting recipes.

- The elephant is the third hostile animal in the world of the game but calling it hostile might be incorrect as it is planned for it to not attack the player once they are in its vicinity, in fact, avoiding conflict it would move away from the player. That would make it tricky for the player to initiate a battle with the elephant as it has a slightly

faster walking speed than the player does. But once they've managed to land the first hit, the real danger begins. The elephant needs only two hits to kill the player and being extremely fast can kill the player easily. As a result of this players may decide to team-up and attack it together and afterwards share the resources. However, killing it isn't impossible and is all about timing, when the elephant attacks the player, it stops in order to hit and it takes it a couple of seconds before it continues to chase the player and it needs a certain amount of time before it launches its next attack. In addition to dropping a lot of meat and some hide upon dying, the elephant also gives the player elephant horn.

2.6 Resources, Items and Crafting

As mentioned before, the main resources in the game are the trees, stone nodes and gold nodes. They're the main building materials for the items that can be crafted in the game. We'll go through the different items and how they can be crafted and used.

- Wood pickaxe - the wooden pickaxe is the first tool that the player can obtain when first starting the game. It's the most basic tool and it is crafted using wood. When equipped, the wooden pickaxe increases the reach of the player slightly and it helps them cut more wood per hit than with just bare hands. It also allows them to mine stone from the stone nodes scattered around the map. The wooden pickaxe also deals a small amount of damage to the players and animals.

- Toolkit - The toolkit is made with wood and stones and it is used to craft other useful tools that the player might need. The player needs to have a tool kit in their inventory in order to craft all tools beside the wooden pickaxe.

- Stone pickaxe - The stone pickaxe is an advanced version of the wooden pickaxe. It is crafted when the player has a toolkit, a wooden pickaxe and enough additional wood and stone in their inventory and this upgrades their wooden pickaxe into a stone one. The stone pickaxe doesn't increase the player's range but it deals slightly more damage to other players and animals and it also allows them to mine gold as well as increases their mining rate.

- Gold pickaxe - The golden pickaxe is the most advanced pickaxe in the game and that the player can obtain. It requires for the player to have a toolkit, a stone pickaxe and enough wood, stone and gold, in order to upgrade their stone pickaxe into a gold one. The gold pickaxe doesn't increase the radius of the player but it deals slightly

more damage to animals and other players than the stone and wooden pickaxe and it allows them to mine more wood, stone and gold per hit, thus saving time.

- Stone sword - The stone sword is the first weapon that the player can craft after starting the game and it's the weakest one. It increases the reach of the player comparing it to bare hands and the pickaxes but the player is unable to gather wood and other materials when equipping the sword. The sword is intended to use against hostile animals and players. To craft it, the player needs a tool kit and enough stone and wood in their inventory.

- Gold sword - The gold sword deals more damage than the stone sword but doesn't increase the radius of the player or their speed. In order to craft it, the player needs to have a tool kit, a stone sword and enough wood, stone and gold in their inventory.

- Ivory sword - The ivory sword is the strongest weapon in the game. It deals the most damage comparing it to all the other tools of the game but it doesn't have any other added benefits. In order to craft it, the player needs to have a toolkit, a golden sword, an elephant husk and enough wood, stone and gold in their inventory.

- Campfire - a necessity during the night cycle of the game when the temperatures drop dangerously low. The campfire requires wood and stone to be crafted and it's placed in the player's inventory from which they can then place it on the map in a chosen location. The player would be able to view where they're about to place the campfire by clicking on it in their inventory when a preview of where the fire is about to be placed will appear. The campfire is not a permanently placed object and it'll run out after a certain amount of time. In addition to providing warmth, it also radiates light but it can also damage the players if they stand directly above it and would continue damaging them for as long as they're standing too close or on top of it. The campfire is also the only place where the player can cook the meat they've obtained by killing animals. When the player is standing near a campfire during the day even if they're not directly on top of it, they'll get damaged continuously.

- Raw/cured animal skin - The hyenas and the elephants drop raw skin upon their death which is directly placed in the inventory of the player who killed them. This animal skin can then be cured next to a fire and used for crafting.

- Water pouch - After the player has managed to receive enough raw animal skin and then cured it next to a fire, they'll be able to craft a water pouch using the tool kit. The water pouch will then greatly expand the player's opportunities for exploring the

world as they'd be able to venture out further from the lakes and still have a water source at their disposal. The water pouch can be filled with water and the player can drink from it regaining some hydration and lowering their body temperature. Additionally, the water pouch can be used to drink several times and the water in it isn't used up all at once. When empty, it'll not provide any benefit to the player. The pouch can be filled up to the maximum amount even if it's not entirely empty.

- Cactus potion - The cactus potion can be easily crafted when the player has enough cacti flesh in their inventory and they're standing near a fire. Having at least one cactus potion at all times is highly recommended as it's the only cure against scorpion venomous sting. As mentioned previously, the scorpion would sting the player and that sting will take away health points from the player until they die, unless they consume cactus potion. The cactus potion would not heal the player but it will counteract the venom in their system and prevent any further drainage of health points.

- Healing salve - the healing salve is the only healing item in the game and the only way for the player to regenerate their health points beside the natural healing that would occur if the player isn't thirsty, hungry, too hot or too cold. The healing salve is crafted using the venomous gland from the scorpion and cactus flesh. When consumed, it'd heal the player a certain amount every couple of seconds until their health reaches the maximum or it has healed them 30% of their health points.

- Wooden crate - the wooden crate is crafted with wood and a few stones and it is used for storage or for transfering items between players. The crate is able to hold any amount of any one recourse or item and if left unattended with its contest, other players would be able to steal the items. The wooden crate is also the only way for the player to "give" or "receive" items to or from other players.

3 ANALYSIS

When building any new piece of software, it is important to first start with understanding what the software needs to do. In addition to this, in this phase we breakdown our system into smaller subsystems and understand what the requirements are.

For this game, we identify user requirements as follows:

- Make a new account – In the final version players should be able to create a new account which would allow the player to see cumulative statistics about their gameplay and unlock skins based on that.

- Play the game

- Login

As per functional requirements, the main requirements are as follows:

- Create new account with given credentials

- Delete account – if the player is not pleased with the game, they may want to delete their account

- Record player's scores – Maintaining an all-time leaderboard is fun and players should be able to see where they stack up against the others

- Ability to ban either temporarily or permanently a person that uses profanity or hate speech

In addition to this, the non-functional requirements of the project in order to determine which attributes the system needs to possess. This is crucial because the game should be played by people of all ages and therefore should limit certain sorts of unethical behavior.

Thus, the main non-functional requirements are:

- The credentials need to be stored securely

- Usernames can't contain profanity or hate speech

- The system should have a way to sanction people that use profanity in the game chat

It is important to note that in the prototype version of the game, not all requirements have been fulfilled. This is intentional as the prototype serves as initial version and in addition to that expanding the software once the main functionality has been implemented is a trivial task.

As authors Dennis, Wixam and Roth point out in “System Analysis and Design” once the requirements for the system are established it’s time to proceed to think of how the actors of the system will interact with it [10]. As actors for Dezertio, we can identify the players as primary actors as well as the administrator as a secondary actor. The administrator’s purpose would be to moderate the in-game chat from time to time and ban those that break the rules. Once we have established who the actors of the system are, it becomes easier to reason how they will use the system. To do this, the UML modeling language is used to specify the use-cases of the system.

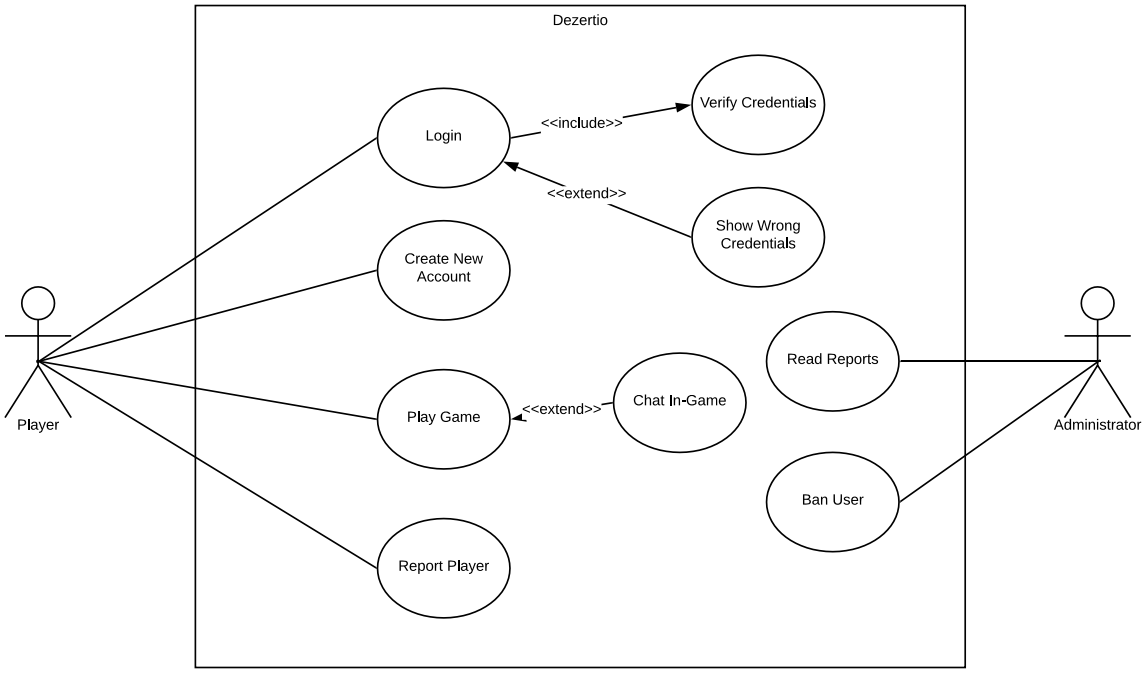

Figure 3: Use-Case Diagram for the System

The above UML diagram gives a clear overview of what functionalities the system needs to support.

Once we have established the use-cases, we can go ahead and describe the DFD of the system. We begin that by first describing the context-diagram which gives us a broad overview of the system in the context of its environment. The context-diagram shows us the major inputs and outputs of the system. As a high-level overview, the players of the game

log in, update the world by taking actions within it as well as can report another player if there is need. The system sends the current state of the world to the player. Adminsitrators have the ability to go over reports made by players and ban a particular player. Thus, we he have the following high-level abstraction of how the system should behave in relation to its external entities.

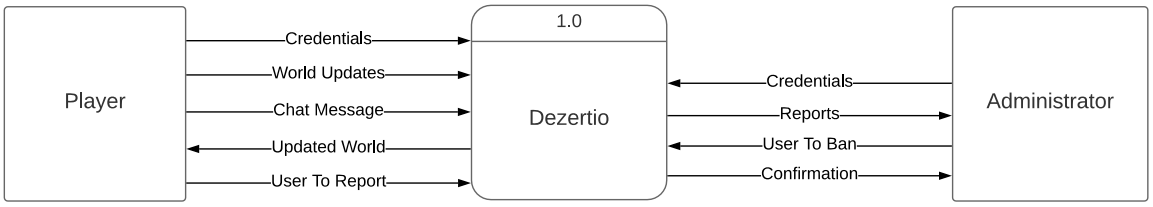

Figure 4: Context Diagram

This, in turn, gives some information and we can further decompose the system into major high-level processes to have even a better overview of the system.

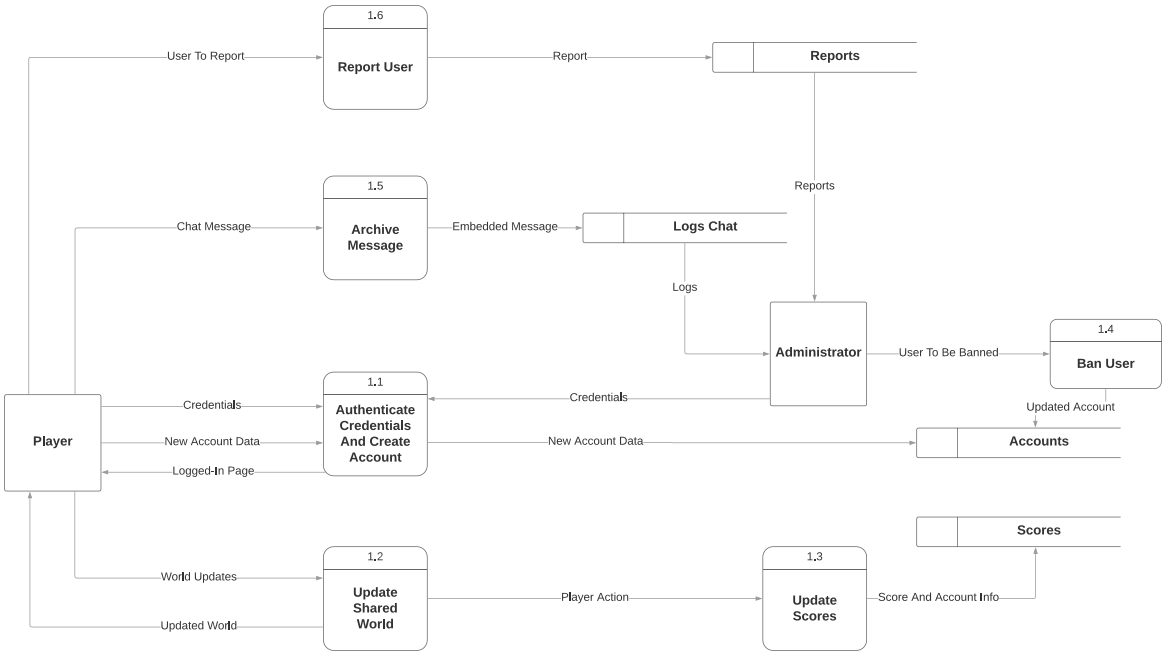

Figure 5: System Decomposed Into Major Processes

4 DESIGN

The first step of creating the system design is identifying the design goals. This is done by prioritizing the qualities of the system that need to be optimized. In most cases, the developer has to choose between what quality is more suitable for the system that they are trying to build and in a lot of cases focusing on a certain quality comes at a cost of another one. Most common trade-offs are functionality vs. usability, efficiency vs. portability, rapid development vs. functionality, etc. This is an important step of the development because it allows for a more flawless software architecture and optimization. For the Dezertio game, the main focus is rapid development, understandability and maintainability.

After the design goals are determined, the system decomposition takes place. It entails breaking down the entire system into smaller components which are then broken down into their building blocks. The factors that are considered during the decomposition of the system as a whole into smaller subsystems are system responsibilities, dependency between subsystems, subsystem mapping to hardware, etc.

Following the defined design goals the whole system is divided into the following components:

- Defining the main systems that the game is built from – the main systems are the client, the server-side and the updating the shared world.

- Defining the relations between the three main systems and how information is shared between them and any changes in one of the systems reflect those changes to the other systems simultaneously.

- Further breakdown of the subsystems and defining their functions at each level and how that information is correlating with the rest of the subsystems.

After we have set the design goals for the project, the next step is to build the architecture. This is an important step of the development because it allows us to systemize the different components that the software, or in our case, the video game uses. Following the Data Flow Diagram (DFD) which we created in the Analysis, we can build a structure chart that will show all the components of code that must be included in the program at a high level, arranged in a hierarchical format that implies sequence (in what order components are invoked), selection (under what condition a module is invoked), and iteration (how often a component is repeated).

Another reason why creating the structure chart is very important prior to implementing the game itself is to ensure that all the code necessary for the game to work as a system will be created and will communicate with the rest of the code as needed. In addition, it would help the development team to measure the approximate time and resources that would be required for the completion of the project before it can go into the testing phase.

The way to build a structure chart is to compose modules which represent lines of code to be written that will perform a single function. This is why it is important to be as detailed as possible and break-down the structure of the project into its building blocks. After the main modules have been created, the next step is to identify the special connections between them by adding loops and conditional lines to represent modules that are repeated or optional (acted upon after a certain trigger or condition is reached).

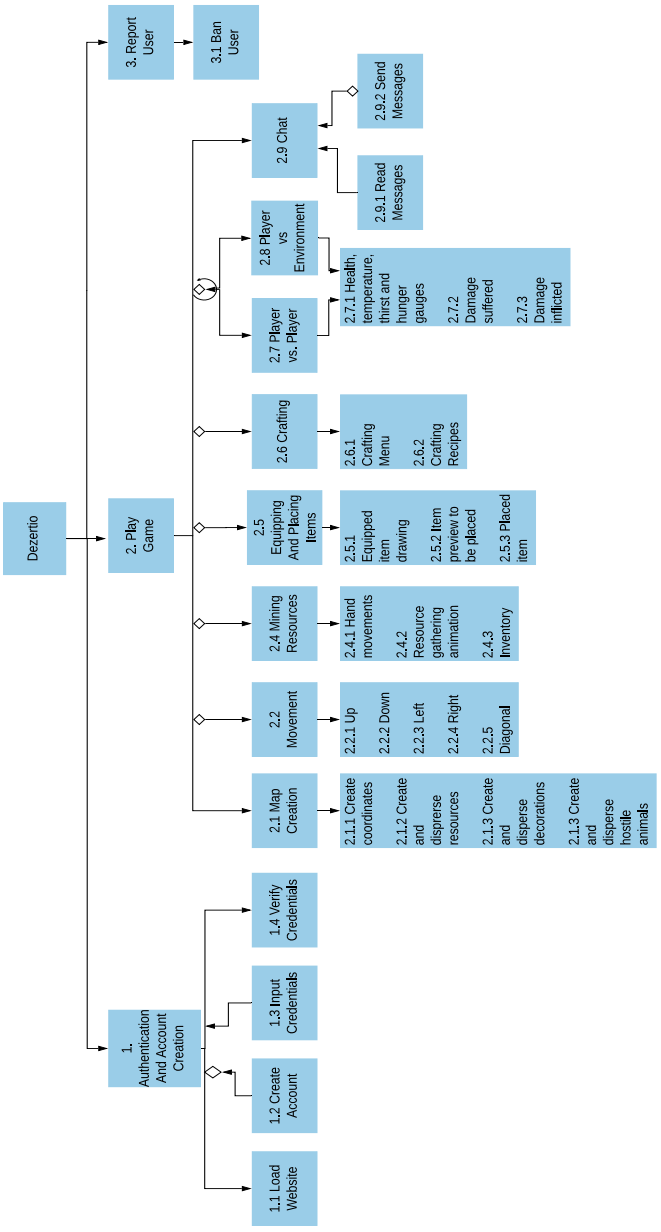

Figure 6: Structure Chart

The main modules we have created for the game are directly taken from the Data Flow Diagram created in the Analysis chapter. The further breakdown of the authentication and account creation is simple. The first step would always be to load the website. The next module is optional – the player can create an account if they have not created one before or they can skip to the log in. After their credentials have been verified, they can move on to playing the game.

The "Play Game" module contains the majority of the code. Breaking this down to its building blocks is much more complex and requires a lot more coupling and conditions. The stand-alone automatic processes are map creation and chat. The "Map Creation" submodule is automatic because no certain action has to take place for this process to occur. This means that as soon as the player begins the game, the map is being generated by invoking smaller subcomponents. In our case the map generation entails the creation of the coordinates for resources, creation and dispersal of the different resources, decorations and hostile animals. These four submodules are the general way that the map will be created each time and displayed for the player.

The "Chat" module on the other hand contains two main submodules one of which is automatic and the other is conditional. Reading messages is automatic and does not require any action to be taken by the player to invoke the messages being displayed on their screen but send message function is conditional because it requires the player to compose their message and send it deliberately.

The "Movement" submodule is conditional and requires a key press from the player. The player can move Up, Down, Left, Right and Diagonally as stated in the graph. The "Mining Resources" submodule is another conditional module which is invoked when the player clicks the mouse over a certain resource while having the right tool equipped which means it's not automatic but rather conditional. Important functions that need to be created in order for the "Mining" resources submodule to be fully functional is the implementation of the equipped item drawing, the resource gathering animation and of course, the inventory itself where the gathered items are placed.

The next submodule in the "Play Game" module is the "Equipping/Placing Items" module. This module is also conditional since it requires a certain action to be taken by the player in order to be invoked and if the player does not take the required action, this module is not going to be invoked at all. The functions pertaining to this submodule are "Equipped Item Drawing", "Item To Be Placed Preview" and "Placed Item".

The “Crafting” submodule is also conditional and the functions pertaining to it are the “Crafting Menu” and the “Crafting Recipes”.

Next, we have two more complex submodules which are coupled together - “Player vs Player” and “Player vs Environment”. These are the two most important submodules in the “Play Game” module due to their importance to the gameplay. In addition to being coupled, the two submodules are also iterating over each other on a loop in certain conditions.

The PvE submodule refers to the changes that naturally happen to the player during the gameplay such as the hunger, thirst an temperature meters being depleted throughout the gameplay as time passes by unless the client does something to replenish them. It also refers to any damage that the player may suffer from being attacked by the hostile animals. Lastly, it takes into account any damage that the player inflicts as well. The same logic is followed for the PvP module but the focus is on damage taken or inflicted by or to other players on the server.

Lastly, on the Structure Chart, we have the “Report User” submodule which is a small but important part of the system to prevent abuse and other types of unwanted activities in a video game played by people of all ages. This is why, having the “Report User” submodule with its single function of banning users from the game is vital.

As we have gone over the “Structure Chart” which decomposes the entire system the way it will be built, it is important to mention other tools which could be helpful with visually representing the workflow of activities within the system which can give us a better idea of the entire system as a whole or a particular complex process that we need to structure. This can be achieved with an Activity Diagram.

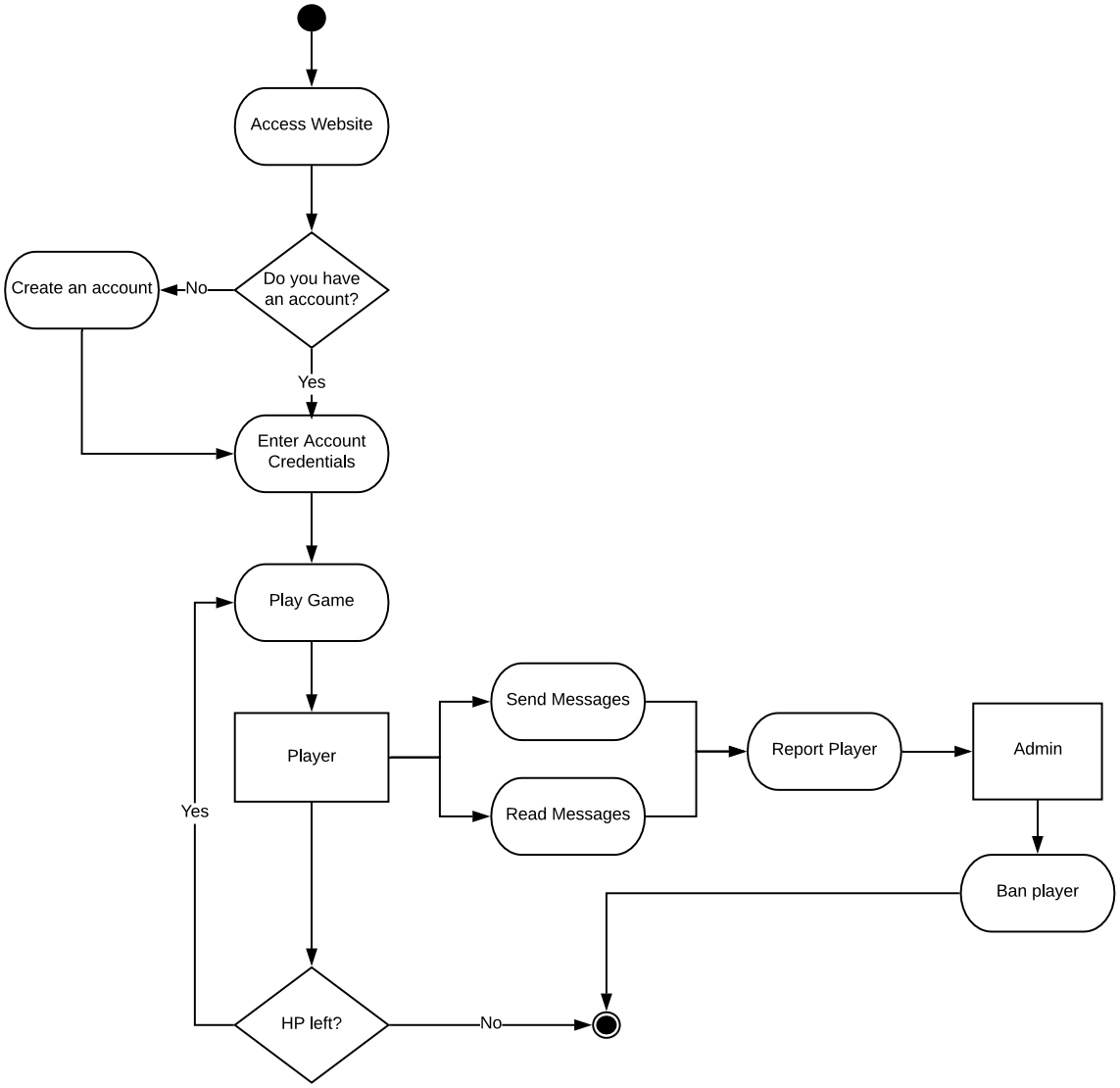

Figure 7: Activity Diagram

With the activity diagram we not only visualize the dynamic nature of the system but we also construct the executable systems by using the forward and reverse engineering techniques. The only downside of the activity diagram is that it doesn't show any messages flowing from one activity to the other but this is also a strength in the sense it allows us to recreate the activity flow of the system which is a usage not available in other types of diagrams.

5 IMPLEMENTATION

When it comes to developing an HTML5 video game, there are a lot of game engines available. However, because Dezertio is a small game, the use of a game engine was not deemed desirable because at the benefit of saving developer time an engine doesn't provide enough fine control over all the aspects of the game as writing it from scratch does. Therefore, the decision was made to write it from scratch in order to utilize the full benefits of complete control over the game mechanics.

In this chapter we go over some broad details regarding the implementation as well as dive deeper into some aspects of the game mechanics.

5.1 Technology Stack

We will now take a look at the technology stack used for developing the game including languages, frameworks and libraries.

5.1.1 Javascript

Javascript is the lingua franca of today's modern Web, it allows us to build sophisticated and responsive web applications and was thus used for the building of Dezertio. The language has many nuances but when used carefully it can be utilized effectively to build video games. The game uses Javascript both on the client-side and on the server-side.

5.1.2 Node.js

On the server-side the game uses the Node run-time environment. The benefit of Node lies in the fact that we can use Javascript on the server-side as well. This in turn saves considerable software development time. Additionally, Node is an excellent choice for many real-time Web applications like browser games. Furthermore, it supports large number of concurrent connections which makes it an ideal choice for a multiplayer game.

5.1.3 Socket.io

Socket.io is a Javascript library based on the WebSocket protocol which provides event-driven bidirectional communication between clients and server. It is used to manage the game state where each client sends their state to the server and in turn, the server broadcasts the current state of the world to each client. The world is constantly updated based on

players' actions and Socket.io allows us to keep each client updated to the newest state of the world.

5.2 Rendering The View

As mentioned in the preceding chapter the application logic is divided between the client and the server. The part of the logic implementing what portion of the terrain is seen at any given time is done on the client side. Each client has variations in viewport size therefore asset scaling must also be addressed.
The world in Dezertio has height and width of 2000 units. To render what the player at any location sees in the world we need to determine the crop position on the terrain. The **sx** and **sy** global variables denote those locations. The below listing shows how that is implemented.

Determining Crop Positions

```
//Computes crop coordinates given the location of the player.
//V.w and V.h are the width and height of the viewport respectively.
//W.w and W.h are the width and the height of the terrain respectively.
procedure crop_calc(playerx, playery)
begin
        if playerx - V.w/2 >= 0
                sx = playerx - V.w/2;
        else sx = 0
        if playerx + V.w/2  > W.w
                sx = W.w - V.w
        if playery - V.h/2 >= 0
                sy = playery - V.h/2
        else sy = 0
        if playery + V.h/2 >= W.h
                sy = W.h - V.h
end
```

The above procedure is invoked every game tick which allows the camera to follow the player as they move through the map. We mainly use world coordinates because it allows us to easily determine which game objects are within the view at any given time. Converting between screen coordinates to game coordinates is accomplished by:

$$pos_x = \frac{C_w}{V_w} \times \tau_x$$

$$pos_y = \frac{C_h}{V_h} \times \tau_y$$

τ_x and τ_y are the world coordinates. C_w and C_h are the canvas width and height.

5.3 Movement And Animation

The basic building block of the codebase is the Vector2D class. It provides methods to effectively work with vectors such as scaling them, rotating them, computing distance to another vector etc. These methods are essential because they are used for collision detection, animation as well as drawing objects correctly relative to the position of the player.

Vector2D Interface

```
interface Vector2D
begin
        add(v);
        print();
        rotate(alpha);
        scale(scalar);
        //Travel from point (x,y) to another point (x', y').
        travel(dist, alpha);
        distTo(v);
end
```

Dezertio doesn't rely on sprite sheet animation therefore the rotate method is utilized for animations like sword swinging and punches. The rotation is an example of a linear transformation that is accomplished by multiplying the rotation matrix with the vector.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

The Entity class is responsible for rendering the correct asset on the screen. Dezertio doesn't implement any fancy lightening effects and instead relies on using different day or night assets. Thus the Entity class provides more general abstraction for directly working with game assets.

Entity interface

```
interface Entity
begin
    //Adjust asset for the day-cycle.
    checkDayCycle();
    setWorldCords(v);
    //Perform adjustments for in-canvas drawing of the assets.
    adjust(adjustX, adjustY, adjustAngle);
    //Render current asset on the screen.
    draw(centeredRot, tipRot);
    //Rotate alpha degrees by using the center as a pivot.
    rotateBasedCenter(alpha);
    //Rotate alpha degrees by using the tip(bottom-center) as a pivot.
    rotateBasedTip(alpha);
end
```

When it comes to animating assets it's of utmost importance that the animations are not choppy. To overcome choppiness due to frame by frame rendering we rely on linear interpolation. That is, given a pair of discrete values **x** and **y** we adjust **x** continually by some percentage of the range between them.

$$x_{new} = x_{old} + (y - x_{old}) \times \frac{p}{100}$$

5.4 Collision Detection

For some objects the player naturally needs to collide with. For the prototype of the game, the player collides with cacti, gold ores and stone ores. To handle collision, we treat assets as if they were circles. We compute the distance between their centers and if it's less than the sum of their radii then a collision has occurred.

In addition to this, for more aesthetic gameplay, we ensure that once the player collides with an object, their new position is adjusted to give the impression of sliding. Constantly updating game state and checking for collisions can be computationally expensive, therefore, we only check for collisions with objects which are within view.

Collision Detection For A New Position Of The Player

```
//Checks if a collision occurs at a new position with coordinates nx, ny
procedure collidesWithAny(nx, ny)
begin
    cx = nx
    cy = ny
    //Assign the radius of the circle to the minimum of width/2 or height/2
- this ensures visually appealing collisions
    if world_width < world_height
        cr = world_width/2
    else
        cr = world_height/2
    //Only need to check collision with objects which are drawn on screen
    for i = 0 to len
        begin
            obj = objectsInDrawingZone[i]
            //Don't collide with other players or mobs
            if obj.mobtype == hyena || obj.mobtype == scorpion || obj.type
== otherplayer
                return FALSE
            distance = dist(cx, cy, obj.cx, obj.cy)
            if distance < cr + obj.cr
                return TRUE
        end
    return FALSE
end
```

Additionally, it would be aesthetically pleasing to enable the player to glide once they collide with an object instead of just stopping. We achieve that by cotinually readjusting the angle of the direction in which the player wants to move.

Sliding Effect When Player Collides

```
while collidesWithAny(nx, ny) == true
begin
    theta += PI/16;
    dx = sin(theta) * d
    dy = cos(theta) * d

    //Readjust nx, ny until no collision occurs
    nx = world_x + dx
    ny = world_y + dy
end
```

5.5 Head-Up Display

The **HUD** shows vital information to the player. It has health, thirst and hunger gauges, inventory, body temperature and crafting menu. In addition to that, it has a scoreboard displaying the name of the 10 highest scoring players. The **HUD** functionality is implemented in the **HUD** class. The draw function inside **HUD** class invokes other functions which step by step draw the head-up display.

The circular health, hunger and thirst gauges are on the top-left side of the screen and are depleted slowly when the player goes lower on hunger, food or thirst. The current HP, thirst and hunger values are read from the **Game** object.

The inventory is drawn at the bottom-center of the screen and it reads the current game state of the player and draws the items possessed and their amount. The instructions for rendering the inventory are trivial. In the global **Game** object we define the number of slots in the inventory as well as the size of each slot in pixels. Based on these values we can calculate the entire width of the inventory and use that to center it. After that, we highlight if a certain slot was clicked on. Then we simply iterate over the **Game.inv** object and render the corresponding image of the possessed item in the center of the slot. The inventory object is defined as a dictionary with key-value pairs with key being the name of the item and the value being an array containing information about the amount of the item and canvas coordinates.

Inventory Object

```
inv: {
        "stonesword": [1, 0, 0],
        "rock": [1000, 0, 0],
        "wood": [400, 0, 0],
        "gold": [300, 0, 0],
        "cactus_flesh" : [40, 0, 0],
};
```

For the last part of the **HUD**, the scoring board relies on the **updateScores** method. The updateScores procedure is ran every 3 seconds and it computes the 10 highest scores. The **Game.other_players** object contains data about all the players in the server. It's a key-value pair with key being the socket id and the value being an array of elements that the client sends to the server. The last element in that array is the points for that particular player. It sorts the keys based on the value of the points and adds a tuple of the name of the player and formatted score to the **Game.scores** array which is then read and rendered on the screen.

Update Scores

```
procedure updateScores
begin
        keys = Keys of Game.other_players
        if keys.length == 0
                return
        Game.scores = []
        //Index of points
        indxP = Game.other_players[keys[0]].length - 1
        //Index of nickname
        indxN = indexOfPoints - 1
        keys.sort based on the value of the points of the player
        for i = 0 to Min(10, keys.length)
                begin
                        n = Game.other_players[keys[i]][indxN]
                        f = formatScore(Game.other_players[keys[i]][indxP]
                        score = [n, f]
                        Game.scores.add(score)
                end
        setTimeout(updateScores, 3000)
end
```

The scores are kept in the **Game.scores** array and then we render them on the top-right position of the screen as they are continually updated.

In addition to drawing gauges, inventory and leaderboard, the **HUD** also implements the functionality of clicking on an inventory item. Items like cactus flesh and meat are edible and they can be consumed when they are clicked on. Also, the campfire is a placeable object and when it's clicked a preview of where it's about to be placed is drawn. In addition to this, mining tools and swords are equipable and are rendered in the hand of the player.

The method which implements the functionality that needs to occur upon a click on an inventory slot is implemented in the **dealWithClick** method in the **HUD** class. It detects where a mouse click has happened on the screen and on which inventory slot. It relies on methods like **isEdible**, **isEquipable** and **isPlaceable** to execute the needed changes. If an item is edible then it directly increases the hunger and thirst gauge by modifying the thirst and hunger variables in the global **Game** object. Similarly for items which are equipable it modifies the global **Game** object which is then read by the **Hand** class and the item is rendered in the hand of the player.

5.6 Crafting

The **CraftingMenu** class implements the functionality of crafting. Items require different amounts of time to craft. Higher tier items require more time to craft. This is intentional as the player is vulnerable during the crafting period. The time to craft an item is calculated naively as frames elapsed. On average, 60 frames are rendered per second.

Crafting Durations

Woodenpickaxe	600
Stonepickaxe	900
Goldpickaxe	1200
Stonesword	900
Goldsword	1600
Placeable Fire	60
Leather Bottle	200

In addition to this higher tier items require more resources in order to be crafted.

Resources Needed To Craft Each Item

Item	Wood	Stone	Gold	Leather
Wood Pickaxe	30	0	0	0
Stone Pickaxe	30	30	0	0
Gold Pickaxe	50	20	20	0
Stone Sword	100	40	0	0
Gold Sword	100	30	40	0
Fire	20	10	0	10
Leather Bottle	20	10	0	10

Only the items which the player has enough resources to craft are displayed in the crafting menu. Each time the update method of the **CraftingMenu** class is executed we read the **Game.recipes** object and check which objects can be crafted.

Determine What Can Be Crafted

```
procedure whatCanICraft
begin
    thingsICanCraft = []
    keys = Keys(Game.Recipes)
    for i = 0 to keys.length
        begin
            item = keys[i]
            if canICraft(item, Game.Recipes[item])
                thingsICanCraft.add(item)
        end
    return thingsICanCraft
end
```

The above procedure relies on the helper methods which check if an item can be crafted. An item can be crafted if the player has enough resources to craft it and in the case of crafting a weapon or a mining tool, it checks if they already possess the lower tier item. After an item has been crafted it is added in the inventory and the corresponding resources are reduced.

Helper Methods To Determine If An Item Can Be Crafted

```
procedure enoughResources(resources)
begin
    w = enoughResource(wood, resources[0])
    r = enoughResource(rock, resources[1])
    g = enoughResource(gold, resources[2])
    l = enoughResource(leather, resources[3])
    if w AND r AND g AND l
        return TRUE
    else
        return FALSE
end
procedure lowerItem(item)
begin
    if item == goldpickaxe
        return stonepickaxe
    else if item == goldsword
        return stonesword
    else
        return woodenpickaxe
end
procedure requiresLowerItem(item)
begin
    if item == goldpickaxe OR item == goldsword OR item == stonepickaxe
        return TRUE
    else
        return FALSE
end
```

Once we have established which items we can craft, we draw them on the bottom-right side of the screen. In addition to this, once crafting has begun we show the progress with percentage. Once a new item has been crafted, the **giveItem** method is called. It adds the new item to the inventory, reduces the amount of resources that were needed to craft it and in case a lower tier item was upgraded adjusts the contents of the inventory accordingly.

Give Crafted Item To Inventory

```
procedure giveItem
begin
    item = this.whatAmICrafting
    amount = 1
    if Game.inv[item] == UNDEFINED
        Game.inv[item] = [amount, 0, 0]
    else
        Game.inv[item][0] += amount

    if requiresLowerItem(item)
        lower = lowerItem(item)
        Game.inv[lower][0] -= 1
        if Game.inv[lower][0] <= 0
            DELETE Game.inv[lower]
    resNeeded = Game.recipes[item]

    ReduceInInventory(wood, resNeeded[0])
    ReduceInInventory(rock, resNeeded[1])
    ReduceInInventory(gold, resNeeded[2])
    ReduceInInventory(gold, resNeeded[3])
End
```

5.7 Multiplayer Support

In order to synchronize the world for all players all clients communicate with the server and exchange data. Clients send proper data bundles in order for the scene to be shared and rendered properly on all other clients. As soon as a client joins the game back-and-forth communication with the server commences. Each client has an associated socket id which uniquely identifies them. As soon as a client joins the server tells them their unique socket id which is important because it identifies them as a unique actor in the world.

Once the client has joined and has been sent their socket id, they are provided with a data bundle denoting the locations of the ores. These are then used on the client to place the corresponding assets on their correct positions. To prevent lag and choppiness in movement the clients sends its own character data on each game tick. In addition to that the server manages the balance of resources in the world such as mobs and regenerates ores at a proportional rate to the number of players.

Initially when a client joins the server, the server tells them their socket id. This ensures that when an event is triggered it can be identified which client triggered the event. Additionally, the server begins sending thirst and food ticks as well as an array of the currently placed fires. The client receives an array of tuples of the coordinates and angle of the fires which is enough to draw them correctly.

Client Joins The Game

```
procedure clientJoinsGame
begin
    socket.emit(socket.id)
    socket.emit(day)
    setTimeout(sendFoodTick, FoodAndThirstRate.food)
    setTimeout(sendThirstTick, FoodAndThirstRate.thirst)
    let arr = []
    for i = 0 to fires.length
        begin
            arr.add([fires[i].x, fires[i].y, fires[i].angle])
        end
    if arr.length > 0
        io.sockets.emit(arr)
end
socket.on("ingame", clientJoinsGame)
```

As mentioned previously as part of the **Player.update** method data is continually sent out from the current client. It contains information about what tool they have equipped if any, if

they are swinging with the tool as well as details about the tool equipped. This in turn means that once the server broadcasts to all the clients they can draw each other correctly.

Client Continually Updates The Server

```
//Inside Player.update method on the client
arr = [Game.player.x, Game.player.y, this.angle, Game.player.xR,
Game.player.yR, Game.player.xL, Game.player.yL, Game.player.equipped,
Game.player.toolSwing, Game.player.toolData, Game.player.alpha,
Game.SocketID, Game.Nickname, Game.player.points]
socket.emit('client_data', arr)
//On the server
procedure processReceivedData(data)
begin
    //players dictionary contains data about all the players
    players[socket.id] = data
end
socket.on('client_data', processReceivedData)
```

The campfires in the game are also managed by the server. A fire lasts for 1.5 minutes. The client implements the animation of the fire burning and radiating light, however, the server is responsible for extinguishing the fires after they have burnt out and updates the clients for any changes of the fires on the map. Additionally, players constantly mine the ores in order to craft items until they are depleted. It is important to regenerate them so that new players get a fair chance of obtaining items and climbing up the leaderboard. The server broadcasts a signal to all clients to regenerate the resources.

To implement PvP functionality it is important to keep track of which player is damaged by which player and by what weapon. Different weapons cause different amounts of damage. The amount of damage that weapons, tools and fists deal is specified in the **Game.WeaponDamage** dictionary.

Damage Caused By Each Item

Tool	**Damage**
Fist	1
Wood Pickaxe	1.5
Stone Pickaxe	1.7
Gold Pickaxe	2
Stone Sword	4
Gold Sword	7

To figure out if a player has damaged another player is simple - at all times **Game.other_players** object contains data about all the other players. Once we ascertain that

the player has initiated a hit which has landed on an object we check the type of the object and if it is another player the server.js is informed in order to reduce the health of the hit player. If a player kills another player, they receive half of their accumulated points.

PvP Damage

```
//Server informs the client that they've been damaged
procedure pvpDamage(data)
begin
    damageDealt  = data[0]
    victim = data[1]
    io.to(victim).emit(whacked, [damageDealt, socket.id]
end
procedure reduceHealthDueToPvP(data)
begin
    Game.HP -= data[0]
    causedBy = data[1]
    if Game.hp <= 0
        Game.hp = 0
        socket.emit(pointsforkill, [Game.socketID, causedBy])
        game.restart()
end
socket.on(whacked, reduceHealthDueToPvP)
```

As a result of PvP damage a player can die and therefore half of their points are given to the player that killed them.

Score Update For A Kill

```
procedure givePointsForKill(data)
begin
    victim = data[0]
    enemy = data[1]
    indxP = players[victim].length - 1
    io.to(enemy).emit(pointsUpdate, players[victim][indxP]/2)
end
```

6 TESTING

Prior to the release of any piece of software, it is necessary to go through extensive testing in order to ensure that there are no system failures or bugs. Releasing a system which has major flaws could cause direct money loss to the company in cases where there are direct purchases involved or indirect loss of revenue by having the reputation and image of the company ruined causing current users to leave or deterring potential ones [10].

When building a free online browser game no direct purchases are involved but it is still important to ensure that there are no system failures or bugs. The main methods of conducting the testing of a piece of software is creating alpha and subsequently beta testing. The alpha stage of the testing is usually conducted by the developers and/or special third-party testers. During this stage a test plan is created which usually has 20 to 30 pages with each page having a specific objective. The objective describes very specific test cases that need to be examined and evaluated. The test objective itself is usually taken from the program's specifications or the source code [10].

It is impossible to test every possible combination of input and situation which is why it is important to have at least 3 test cases for each objective, depending on the complexity of the objective itself.

Thus, there are generally 4 stages of tests:

- Unit tests – unit tests focus on an individual program module that performs a specific function. In the case of Dezertio, a unit test can be hand motion of the avatar. Usually the unit testing is conducted by the developer and it is on program level.

- Integration testing – the integration testing expands on the unit testing in the sense that it tests two or more program modules that are related to each other. There are four approaches to integration testing – user interface testing, use-scenario testing, data flow testing and system interface testing. In the case of Dezertio, an integration testing can be conducted for the 3 functions – hand movement of the player, equipping an item (pickaxe) and mining a resource. This test is also usually conducted by the developer.

- System test – the system test is similar to the integration testing but on a much broader scope. It tests whether all or most modules work together as a system without errors and if it meets the initially set-up requirements for the project.

- Acceptance test – the acceptance test elaborates on the system test but it focuses less on finding system errors and more on ensuring that all requirements initially set are met.

After the acceptance test is completed, this concludes the alpha testing of the system and we can proceed with the beta testing which would involve using the system with real data and closely monitoring it for errors. During this phase of testing, an initial, beta-version, of the system can be released to the end-users either to initially selected individuals or openly to all users.

Dezertio was tested by friends and family. With that in mind, for we've created a few sample tests documentation that include the results of the initial unit and integration testing.

In this test, the objective is to verify the ability of the player to move through the map using the ASDW keys and that pressing A+W and D+W keys on the keyboard will result in diagonal movement. Using non-numerical values "True" and "False" we verify if the test case is successful or not. As we can see in the diagram, the test has failed because the #12e test case gave the result "False" which means that during the test, the avatar was not able to move diagonally. Therefore, the bug in the code was fixed and the same test was done again.

Test plan

Porgram ID: Dezert.io Version: 1.0.0

Tester: Irhad Date: 15/02/2020

Test ID: #12-1 Requirements: Verify movements.

Objective: Test if the player's movement on the map is valid with keys ASDW

Test cases:			Results:
1)	#12a-1	Move up	True
2)	#12b -1	Move down	True
3)	#12c-1	Move right	True
4)	#12d-1	Move left	True
5)	#12e-1	Move diagonally	True

Similar to this one, other unit tests have been completed and were used as a reference which allowed for the fixing of various bugs and glitches in the system. Some of these unit tests include:

- Verifying hunger decrease – the objective with this test was that the hunger gauge decreases as time passes
- Verifying temperature increase during day-time
- Verifying temperature decrease during night-time
- Verifying thirst decrease
- Verifying hand movement

With the successful conclusion of the unit testing, we can move on to planning the integration testing. As mentioned previously, the integration tests need to expand on the unit tests by combining one or more functions together. Examples of the main integration tests that need to be planned and conducted are:

- Verifying the player's ability to mine resources – The objective of this test would be to ensure that the player can equip a tool and click on a resource and extract a certain amount of said resource

- Verifying the player's ability to craft items – This test has to be conducted for each separate item that can be crafted in the game and the objective is to ensure that when the player has sufficient resources, the item's icon would appear in the crafting menu and when clicked on it will take the amount of time specified in the code and then appear in the player's inventory

- Verifying the player's ability to cause the specified amount of damage to a hostile animal with different types of weapons

- Verifying the player's ability to consume food and get its benefits (increased hunger, thirst and temperature)

- Verifying the player's ability to place fires

- Verifying that the fire extinguishes after some and that it damages the player if they are standing directly on top of it

Successfully completing all major integration tests, we can proceed with the system testing at which stage other people will be included. The system testing would not include pre-planned test cases which need to be verified with a certain value but rather the people involved in the system test would run the game and play it as a complete system while purposefully looking for bugs or glitches that need to be addressed. In the case of Dezertio, friends and family will join the game and be asked to go over the entire map, craft items, place fires, fight hostile animals and eventually each other and after that fill out a document which will include their impressions of the game focusing on any negative experience they might have had while playing it so it can be addressed prior to the official release of the game.After satisfying every aspect of the alpha testing of the game, an official beta-version will be released and become available to all players that want to join who will be then be able to express their opinions positive or negative. These opinions and suggestions will be taken into account before the official release of the completed game.

7 CONCLUSION

The author hopes that this project has been indicative of how building a video game has been a creative pursuit, utilizing a diverse set of skills and that the Web is a very able platform offering an excellent way to reach vast numbers of players. In addition to that, there are several areas of improvement of the prototype. The main aspects for the improvement of the prototype are as follows:

- Movement of Mobs – the prototype has a lot to be desired in terms of the movement of mobs. There are some known bugs associated with this and making mobs mimic the natural organic movement of hostile animals is important

- Optimization – A big portion of the code hasn't been written with optimization in mind. There are a lot of areas to improve in terms of making the code more efficient and one of them is to use another library instead of socket.io to reduce overhead in sending data back and forth. A binary way of doing it is possible and will significantly improve performance and memory usage. As a result of this the game will be able to support a large number of concurrent players

- Animations and Better Art – There are very few games that offer mediocre game art and are successful. Improvements in this area are crucial and they can either make or break a game.

- In-game chat: The prototype of the game hasn't implemented the ability to chat in-game. This is important in terms of players coordinating their movement in the world and either cooperating or forming teams

- Code-Readability: The development of the prototype was a solo project and thus not much value was given to code-readability. In addition to this, there is repetition of code in multiple places. For a solo project this is forgivable but for development of a game in a team, code-readability will enable new developers to be able to quickly get on board and reduce duplicated effort

8 DALJŠI POVZETEK V SLOVENSKEM JEZIKU

Igre so že tisočletja del človeške zgodovine, s pojavom tehnologije pa so vstopile tudi v digitalni svet. Igričarska industrija je zrasla v milijardo dolarjev vredno panogo in postala del vsakdana ljudi vseh starosti širom sveta. To dejstvo, okrepljeno z neomejenimi možnostmi ustvarjanja interaktivnih izkušenj, je pritegnilo zanimanje mnogih razvijalcev.
V tej nalogi podrobneje preučimo življenjski cikel razvoja sistemov spletnih preživetvenih iger za več igralcev. Ta postopek se vedno začne z načrtovanjem funkcij prototipa igre, ki služil tudi kot osnova za vse prihodnje posodobitve iste igre. Naslednji korak je poglobljena analiza programske opreme in zahtev njenih zmogljivosti, pri čemer sistem razdelimo na komponente in ugotavljamo, kako bodo med seboj komunicirale in delovale kot en sam sistem.

Sestavni del izdelave katerega koli dela programske opreme je, da si zastavimo prave cilje zasnove igre, take, ki najbolj ustrezajo programski opremi, ki jo poskušamo izdelati. Ko je sistemska arhitektura opredeljena, nadaljujemo z razpravo o izvedbi glavnih gradnikov programske opreme. Ko je prototip izdelan, pride na vrsto testiranje, ki je ključno, saj lahko razkrije velike napake v arhitekturi, zasnovi in načinu izvedbe. Testiranje je potrebno, če želimo izdelati sistem brez večjih napak.

Dezertio je v osnovi napisan z uporabo Javascripta tako na strani odjemalca kot na strani strežnika, kar je omogočeno z uporabo okolja Node run-time. Poleg tega za dvosmerno komunikacijo med odjemalcem in strežnikom uporabljamo programsko knjižnico »Socket.io«. Uporaba igralnega pogona ni bila zaželena, saj pogon, čeprav razvijalcu prihrani čas, ne zagotavlja dovolj natančnega nadzora nad vsemi vidiki igre, kot če jo ta sprogramira čisto od začetka. Poleg tega želimo za prototip prikazati samo najpomembnejše mehanike iger.

Igro so močno navdahnile igre, kot so »Don’t Starve Together«, »Terraria«, »Minecraft« itd. Od teh iger smo si izposodili obstoječo osnovno mehaniko iger, kot so crafting (gradnja), PvP (igralec proti igralcu), mining (izkopavanje), dodajamo pa tudi nove. Igro lahko igramo neposredno v brskalniku, v katerem se nemudoma naloži, zaradi česar je dostopna širokemu krogu igralcev.

Za izdelavo preživetvene igre smo se odločili, ker je ta zvrst postala zelo razširjena v vseh vrstah medijev in razvedrila, še posebej pa v video igrah. To je predvsem posledica človeške prirojene želje po opazovanju in doživljanju nevarnih situacij, ki bi lahko bile resnične, hkrati pa tudi izzivov, ki jih ljudje v vsakdanjem življenju običajno ne izkusijo. Drug

zanimiv razlog, zakaj je preživetveni žanr postal sestavni del video iger, pa je njegova vsestranskost v igralni mehaniki, ki jo lahko uporabimo in z njo ustvarimo številne podžanre. Igra je umeščena v puščavsko okolje, za katero je znano, da je eno najbolj ogrožajočih okolij na svetu. Igralec mora za preživetje nabrati težko dostopne dobrine, iz katerih izdeluje predmete, ki mu pomagajo preživeti. To opravilo pa je še nadalje oteženo zaradi nenehnega izmenjevanja žgoče vročine dneva in nočnega mraza. Razpoloženje, ki ga poskuša igra ustvariti, je večna nevarnost, ki lahko preži za vsakim vogalom, pa te nevarnosti ne ustvarjajo le živali, ki poskušajo ubiti igralčev lik, temveč tudi drugi igralci.

Končno pa tudi prototip igre ne predvideva vseh funkcij, denimo klepeta v igri, ki bi igralcem omogočil sodelovanje pri lovu na združbe, ki so premočne, da bi se jih lahko lotil igralec sam, ali uporabe lesenega zaboja, ki bi omogočal trgovanje z dobrinami. Poleg tega je mogoče izvedbo precej izboljšati, na primer z refaktorizacijo kode, da ta postane bolj modularna in lažje berljiva, pa tudi z optimizacijo igre, da lahko podpira veliko število sočasnih igralcev. Kar zadeva optimizacijo, »Socket.io« ustvarja nepotrebne višje režijske stroške in ga lahko nadomestimo s knjižnico »ws«, ki je glede na merila uspešnosti učinkovitejša in se lahko uporablja za nadaljnje povečanje števila sočasnih igralcev. Poleg tega lahko velikost podatkovnih paketov, ki jih izmenjujejo odjemalci in strežnik, dodatno zmanjšamo z uporabo formata izmenjave podatkov »msgpack« namesto JSON.

9 REFERENCES

[1] T. Berners-Lee, *"Information management: A proposal"* https://www.w3.org/History/1989/proposal.html Published: March 1989, May 1990 [Accessed: August 2020]

[2] MDN Contributors, *"HTML5"* https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5 [Accessed: April 2020]

[3] P. Piccione, *"In Search of the Meaning of Senet",* https://www.academia.edu/8169696/In_Search_of_the_Meaning_of_Senet Published: July/August 1920 [Accessed: May 22, 2020]

[4] Author unknown, *"The first video game?"* https://www.bnl.gov/about/history/firstvideo.php Published: Unknown [Accessed: August 2020]

[5] D. Winter, *"Magnavox Odyssey: First home video game console"* http://www.pong-story.com/odyssey.htm Published: 1996 [Accessed: 2020]

[6] T. Dobrilova, *"How much is the gaming industry worth?"* https://techjury.net/stats-about/gaming-industry-worth/ Published: April, 4 2019 [Accessed: August 2020]

[7] Internet World Stats, "*World internet usage and population statistics June 2019, updated"* https://www.internetworldstats.com/stats.htm Published: June 2019 [Accessed: August 2020]

[8] C. Pairitz Morris, *"The demographics of video gaming"* https://www.earnest.com/blog/the-demographics-of-video-gaming/ Published: April 19. 2018 [Accessed: August 2020]

[9] K. Kienast, *"Online gaming among children in the United Kingdom (UK) 2017-2018, by age"* https://www.statista.com/statistics/274427/online-gaming-among-children-in-the-uk-by-age-group/ Published: July 24, 2019 [Accessed: August 2020]

[10] A. Dennis, B. H. Wixom, R. M. Roth, *"System Analysis and Design".* Fifth edition, John Wiley & Sons, Inc, Published: 2012 [Accessed: August 2020]