TUSG: The Untitled Sandbox Game

Abhi Jouhal & Stone Darrow
ajouhal@monmouthcollege.edu
sdarrow@monmouthcollege.edu
Monmouth College
Computer Science

December 9, 2016

Table of Contents

Abstract

TUSG is an experimental video game that explores the user involvement and how that shapes the game in return. This game pushes the players to be creative by providing them with the tools to build and manipulate their environment. It will be a refreshing entry into the sandbox-shooter genre.

TUSG allows users to interact with the world as well as other players inside the game. This means that the player can create a wide variety of structures and go into battles with other players. The game creates a competitive experience for people who are into that, while also providing a relaxing mining-building environment for people who don't want to stress on being competitive.

TUSG also provides a safe and engaging social climate for the users to interact with each other while they play the game.

1 Introduction

TUSG's main purpose is to allow the players to mold the game in their own ways. Given the scope of this project, that can only be extended so much. Hence, the players will have the ability to build and break the environment around them inside the game. They will also be able to interact with other players in various ways.

1.1 The Gaming Industry

The video game industry has been growing massive with the easy availability of performance heavy computers. There has been an upshoot in the desire to produce more realistic graphics. The industry, as a result, has been moving heavily towards producing unfinished, dull, uninspired projects that offer pretty much the same gameplay and story but with newer skins slapped on. The EA sports games are the best examples for this. They release games like Fifa and NBA 2k every year or so with very minimal changes. Other sketchy tactics like selling half the game in DLCs, releasing unfinished products when promised otherwise like in the case of *No Man's Sky*, and making the games necessarily pay-to-win, and so on have turned off huge player bases. These issues shows up less in the indie game scene. Indie and art-house games sometimes tend to push the boundaries and bring new, more innovative ideas to the market.

Wanting to go with that flow, we decided to make our contribution to the indie game industry. We were heavily inspired by the concepts of the film Ready Player One. The film is about a gaming system called the Oasis that is necessarily a virtual world. The players are able to venture inside the virtual world using any types of characters they desire to be. The world itself seems real with many exciting things (mini-game style stuff) to do there. The game is set in a distant future but we wanted to try and emulate a similar experience with the technology available today.

Given that we are just two students who have no game development experience, we had to scale the project down quite significantly from the original film.

1.2 TUSG

A major problem we see with games nowadays is the replay value. The freedom a user has in the gameplay is heavily limited. With our game we would like to solve these issues, and allow the users to freely interact with their environment as well as with other users. Hence, the game that we decided to make is a web browser based, 2-d game. It still remains a multiplayer experience that allows players to build structures and attack one another. In terms of categories, it would fall under the shooter-sandbox genre. We believe something with this kind of innovation would keep a user coming back to play, as they could do something nobody else has already done in the game. Since the opportunities in the game are endless, people could get into unending battles to be the first to make something extremely significant within a world. Competition in video games is typically the most efficient way to keep users engaged with one another, and to keep them coming back for more. We see this with first person shooters quite often, even though people get mad when they perform poorly they will continue to play to become better than their opponent. As previously mentioned, our game is a 2D shooter-sandbox, so we want to introduce this element of competition into a unique new game experience. There are no real big name 2D shooters, but we could make a slight comparison from our game to *Territory Wars Online*. It is a turn based shooter, where the amount of distance traveled for a character is limited every turn. We want to avoid the turn based idea, and have the user roam freely while still being able to modify the environment around them. There is no specific way to win one of these battles other than to collect loot from the dead body. Overall the fighting system sounds overall like a mixture of *Territory Wars* and *Fortnite*.

1.3 Gameplay

All of the aspects that we described in the *Gaming Industry* section are not completely what define TUSG. While these are the main aspects that we believe to make the game unique there are many other features that the game will include. Such basic features include attacking other players, item upgrading, and player movements.

1.3.1 Mining and building

Building and destroying the environment is the fundamental operation of TUSG. Players have the ability to break any block in the world, and collect to build with themselves. For example if a player would like to build an ideal house then they would likely make the ground flat by destroying the ground/grass blocks. They would then destroy trees, collect wood, and finally replace that would where they wish to make their house.

1.3.2 Combat

The combination of these basic features and advanced features allow the player to have a unique experience unlike any other game. Even though things such as player movement and attacking sounds extremely simple we plan to make these unique in their own way. Player movement includes vertical and horizontal movement as well as teleportation. Players will be given the ability to create teleporters that they can place where they would like, which would make fast travel much more convenient. These teleporters can be used in a wide variety of other ways, like setting up traps for other players or any other creative things a player would like to do. Setting up traps is a way to “surprise” attack another player within the server. When attacking other players you have a wide variety of ways to do so, including: basic melee, using the environment, and using weapons. Setting traps was our idea of environmental attacking, and to elaborate you will be able to knock people off of ledges to take fall damage or possibly fall into lava. Basic melee is self explanatory, as every player will be able to hit another player using their fists to deal minimal damage to others. There will be premade weapons in the base game as well used to attack players from a distance that deal significant damage to others. This overall fighting system seems standard of most games, but we intend for the building aspect of the game to change the feel of fighting completely. All of these aspects define the basics of TUSG.

1.3.3 Combining the two

On a more advanced level, we want to implement this building into the fighting system for a better and original fighting experience. The idea behind fights is to let players protect themselves and give themselves tactical advantages against other players. Users can build walls to protect from projectiles or build ramps to gain high ground on somebody to shoot over the wall. Weather a person likes to play safe in fights or heavily attack, both parties can play to their style and strengths. This eliminates “spam” fighting in games like *Minecraft* where a player just smashes the attack button, weather it’s melee or projectile, and one player luckily comes out the winner. TUSG stresses strategy and talent over luck in its fighting system, and we believe the building enforces these ideas.

1.4 Multiplayer Experience

TUSG is meant to be a fun game played with friends, which has been implied by the gameplay section before. We are designing the game to be played over a web browser, and played through a router. All users will need to connect to the same router in order to play in the same world together. Connecting to the same router allows for all players to use the same server, and run with reliable speed.

In order for our game to work we will need to run it through a router with a static IP address. IP stands for Internet protocol, or to put it simpler it is an address to send our game information, which allows other computers to reliably find the server when connecting to the router [1]. This static IP gives our game a unique IP from any other web browser page, so computers can only connect to TUSG using this IP on their router. This means that players can only play the game through this router, which allows the game to run more consistently than through many others.

When deciding to create the most reliable gaming experience possible for the user, we decided that a web browser game was the most reliable source. A web browser game is a game played over the world wide web and holds a strong focus on social interaction. Social interaction is important with our game, as it will contain a chat system to type messages to players currently within the server. If there is a user that is not online it will not be sent to that player. This chat system will also display specific world interactions that are deemed very important to all online players as well. One example of an important event would be a player being killed by another. This gives other active players a sense of what users they should be careful of when playing in the world.

In order to have a social interaction between users we want every player to have their own account information. This includes a username and password every time they attempt to play TUSG. This information will be stored in a database and will not be lost, so every player's progress is maintained whenever they decide to stop playing. In a game where player progress is so dependent on the items they collect, we understand that this feature is absolutely essential for TUSG to run.

Speaking of items being held by a user, another separate database is needed for this. The items and blocks mined will be stored in the users inventory, and we need this to be specified. Without this specific database no user would be able to collect items, or mine them making the game useless.

2 Project Description and Analysis

As mentioned earlier, this game will be a multiplayer, 2d, shooter-sandbox. Let's expand upon the terms "multiplayer" and "shooter-sandbox" in the context of TUSG. Our aim is to create a dynamic game system that grows with the players. We want the players to have the ability to build their own playable assets. Games like *Minecraft* and *Terraria* allow the players to do that to some degree. However, going beyond that, we want the players to be able to introduce new mechanics into the game as well. Hence, allowing the players to take the game in any reasonable(they shouldn't be able to just break the game) direction they would want to. This would be somewhat similar to the mods that programmers make for certain games. However, this will be accessible and relatively easy to use for everyone that gets the game, not just people with a programming background.

2.1 Features

The *gameplay* section above explains how the game will operate. These operations require certain features to be included into the game in order for everything to work properly. These features are explained in detail below:

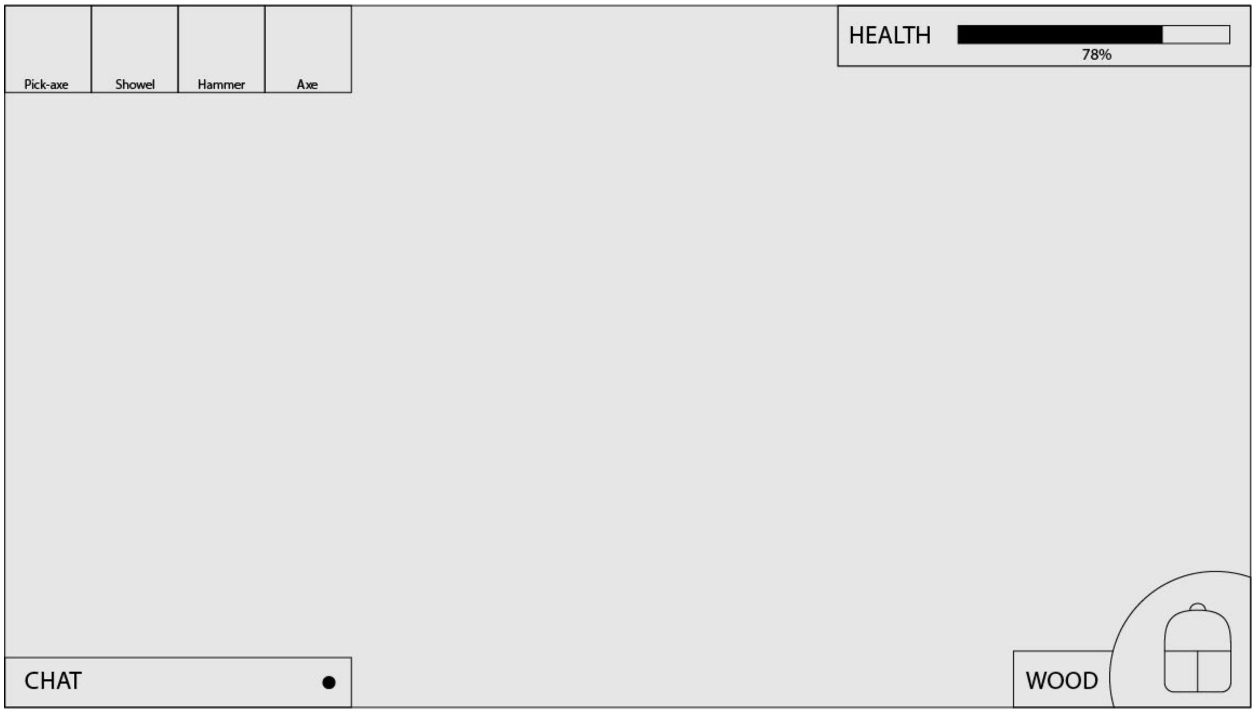


Figure 1: General user Interface

2.1.1 Mining

The term mining in sandbox games means to destroy a certain structure in the game environment and obtain some materials from that structure. For example, if a player comes across a tree, they can use their provided tool to break it down and obtain wood from it.

This infrastructure system provides players with an outside the box experience, where they can do whatever they please to do. The number of 2D games that also function as "open world" are extremely seldom. Open world is in quotes, because there are boundaries that mark players from traveling to far left, right, up, and down. We believe TUSG and its ability to interact with this open world allows for the user to have a new gaming experience. While there are very popular games that feature this mining mechanic such as *Minecraft* and *Terraria*, this game should stand out a little more. The mining and building system will be less complex than that of the other two titles, where there are not different varieties of tools for harvest. For example in *Minecraft* you must craft a wooden pickaxe to mine stone, and then use stone to make a stone pickaxe and so on. *Terraria* has tools that can not mine certain materials, and some that mine everything in the game. TUSG will feature preset tools that every user will have, that mine every object at the same rate.

This may seem like it takes the challenge out of the game, but the primary focus is to allow players the ability to build what they want when they want to. This system allows for users to use the blocks they want to for their structures quicker than having to grind a long while for them.

How does this differentiate TUSG from these other sandbox games? *Terraria* and *Minecraft* focus heavily on adaptation and survivability over creativity (in their respective survival game modes). Where these aspects do apply to TUSG they are much less prevalent than our users creativity spectrum. While players do have to be aware of others attempting to attack them, they more than often will be collecting blocks to craft their own masterpieces. Anything from homes to secret underground bases the possibilities will be endless (until they reach the world boundaries of course).

2.1.2 Materials

While mining, the players destroy a given structure. After that has happened, the player gets a certain amount of a particular material from the said structure. Going with the already given example, destroying a tree will give the player a certain amount of wood that can then be used for the building process.

The materials/blocks in this game are the bread and butter behind everything. All blocks are used to mine, create, and traverse across. These blocks come in a wide variety of different assortments and categories. The categories include ground, wood, metals, and rocks. Within these categories will be sub-categories that are based on their category, but have their own uniqueness. For example, ground could contain a dirt, grass, sand, or gravel blocks that can be mined and used to build.

2.1.3 Building and Crafting

The materials collected from mining can then be used to build different types of structures. The building process will work somewhat like lego. There will be certain basic pre-formed object types available. The player can use them in various different combinations to create elaborate structures within the confines of the environment.

To elaborate on this a user should be able to use two wood blocks to create a door and there will be many more crafting options. Crafting will play a key role in allowing users to unleash their creativity to a different degree than just using blocks. Items that are crafted do not have to be completely block shaped and could be more rectangular for items like the door, or possible barricades to use when fighting.

2.1.4 Health Bar

The health bar within our user interface is a clear cut indicator of how much health you character currently possesses. The bar will lower when injured by another user, and can be regained by sleeping in beds. If the players health is completely depleted, then the user will have a short spawn delay, and will recover from the bed that they slept in.

2.1.5 Tools

Every player will have four tools that they carry at all times that do not take up backpack item slots. Each of the four tools have a specific block category that they mine efficiently, and 3 other categories that they much less efficiently. The four categories of materials (wood, rocks, ground, and metals) each directly correlate to the tools. These four combinations are:

- Pickaxes mine rocks
- Axes mine wood
- Shovels mine ground blocks
- Hammers mine metallic ores

2.1.6 Backpack

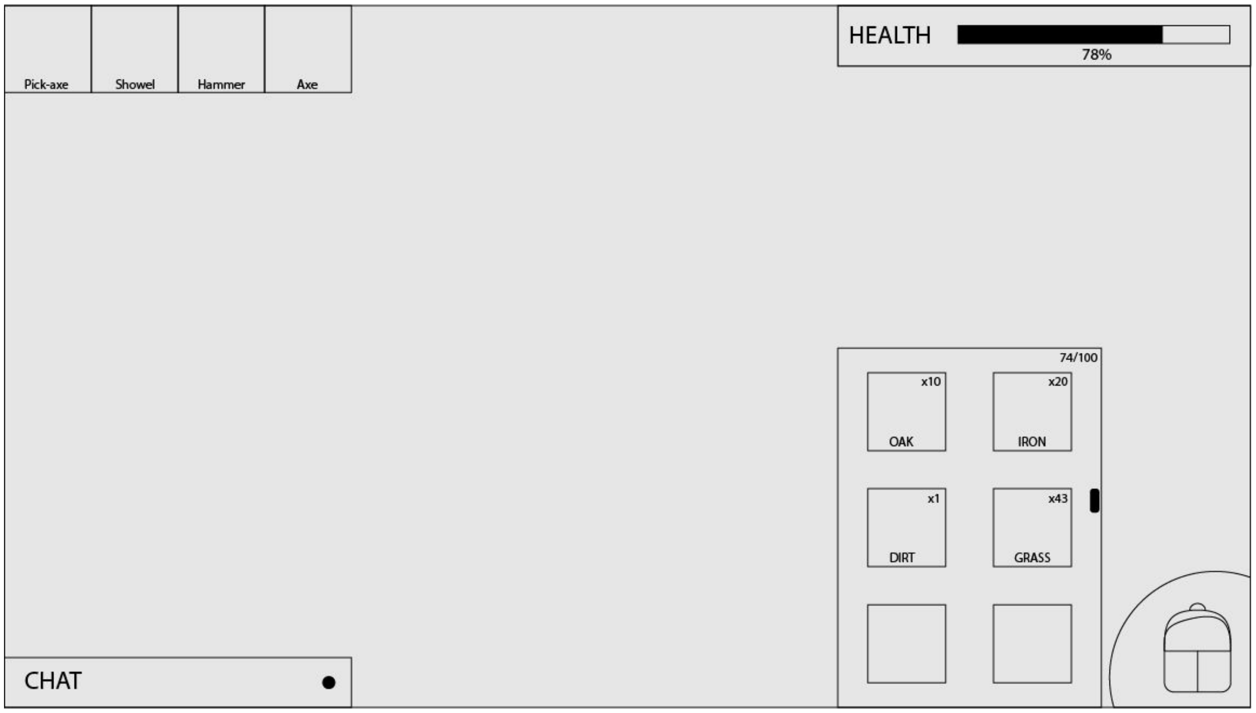


Figure 2: Inventory (backpack) Management

The backpack that a user carries can hold up to 100 different blocks/materials of the users choice. The backpack does not account for the tools that a user automatically receives upon entering a game. When a user has more than one of the same blocks or materials then the item will stack in the backpack and display how many of that item the user has. Whenever an item is placed inside the bag the capacity number is increased and is decreased when an item is removed.

2.1.7 Chat

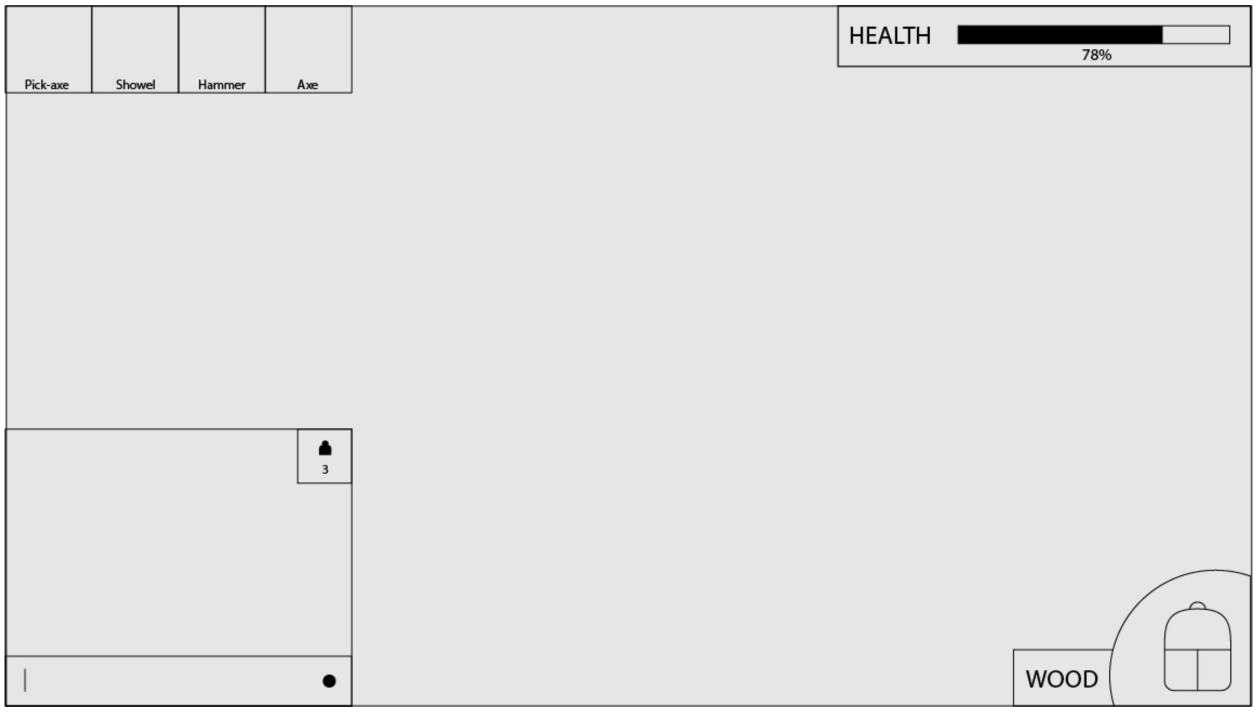


Figure 3: Chat box opened

Every player within the server will have the ability to type messages to other players that are currently in the server. This chat display will be on the bottom left of the user interface and will alert a user when a message is sent, or when an event of significant importance is reported.

Social abilities in TUSG are very important to us, so we are putting a good amount of focus into an ability to chat with others. The message will only be sent to active players in the world and will not be shown to any signed up user that is offline, because we do not need any redundant information being sent.

Why is chatting important over the same router? Some users may play the game in the same room, but this can be played from different rooms in the same building. Hence, the reason this is stressed with high importance. This gives players outside the room a better experience, and makes them feel included with others playing the game. Most video games seem to be better received when there is a form of good open communication amongst peers.

The idea behind the chat system is to use Websockets and socket.io to allow communication from a client to server. WebSockets is a protocol that allows a bilateral synchronous exchange between a client and a server [2]. Socket.io is a library that is based off of WebSockets to make things easier.

2.1.8 User Logins and Accounts

Every user that plays the game will have to sign-up and create their own account. This feature gives all players that access the world the ability to maintain progress they have made weather it was on home they built, or all the items they had saved up. When a player decides to get off of the game then their account will become idle, and will no longer display their character in the world. Being offline will also restrict the player from receiving messages until they are signed back into the world again.

2.1.9 Workshop Features

The workshop is an experimental feature that we want to try to implement. This feature will allow players to create their own unique game assets and mechanics. For example, if a player wishes to, they could give their characters the ability to fly. The workshop will have some restrictions so the player don't just break the game. This will be the most challenging feature for us to implement, so it's mostly just experimental stuff for when/if we have everything else done.

2.1.10 Weapons and Traps

In section 1.3.2 the idea behind projectile, melee, and traps were introduced briefly. All three of these forms of attack have their own strengths and weaknesses to them, and can be utilized however a player intends. Melee weapons do much more damage than bare hands, but all guns will hit for two different types of damage depending on the gun. Traps are different and can vary in the amounts of damage they can deal.

Melee is the most simple form of attack where the user can walk up to another user and deal damage to them. This form of attack can do great damage with swords, but runs a high risk of attempting to get close to another player.

Projectiles don't hit as hard as melees, but they force other players to keep their distance. Guns will shoot at two different speeds, one being fast and the other being slow. Faster shooting does slightly less damage where slow shooting hits harder. These are quite balanced and is more so up to a user preference in weapon type.

Traps are more difficult to explain, because they can do a wide variety of damages to users. Players have the ability to craft traps that can do 25 damage to another player if they step on it. On the other hand, players can use their creativity and use the environment to create makeshift traps. This makes it difficult to discuss the damage being dealt, because players can trick people to fall off things and take damage, or possibly fall in lava, which deals significant damage.

2.2 Ethics

With every piece of software that is released for public use, there are always some ethical questions that come up. These issues are even more prevalent and worth concern in cases where online interactions of users is involved. Some of those issues are explored in this section.

2.2.1 Toxic Users

A lot of people like to be very toxic whenever they are behind a keyboard. This toxicity can include comments directed at people, general trolling, profanity, threats, etc. Given that we are opening this game up to all age groups, it is not wise to have kids be subject to that sort of behaviour.

To tackle this issue, we will add default profanity filters and reporting mechanisms for abusive chat. With those features in play, the chat will filter out the illicit/abusive words. The specific user can then be reported and possibly be banned from the game, based on how bad they act in the game.

2.2.2 Users with malicious intent

With user logins involved, there can be many different types of people online who are interested in others' data. These people can include hackers who steal data and sell it, sexual predators who would like to lure kids into something malicious.

These kinds of issues are not too big for our project because we don't really ask for any such data that can be used for ill purposes. We will not be asking users for any personal data. As far as predators are concerned, we will give a warning to not give out any personal information to strangers every time someone starts the game.

2.2.3 Addiction

Another big issue with gaming (and social media software as well) is that users can get addicted to the games. They can end up spending long amounts of time without leaving their gaming setups. This can be a very serious issue.

Our intent is not to promote any such behaviour as we believe in maintaining a healthy balance of everything in life. To tackle the issue, we will issue warnings and reminders to take breaks if a player has been online for more than 4 hours.

3 Foundations

This section goes over the necessary elements required to implement the aforementioned features into the game. We will also discuss how hard a certain feature may be to implement and, as a result, the plausibility of them getting implemented.

3.1 Godot

Godot is a "free and open source community-driven 2D and 3D game engine" [3]. It is very powerful, yet user friendly. The Godot documentation on their website explains the engine as:

> Godot Engine is a feature-packed, cross-platform game engine to create 2D and 3D games from a unified interface. It provides a comprehensive set of common tools, so users can focus on making games without having to reinvent the wheel. Games can be exported in one click to a number of platforms, including the major desktop platforms (Linux, macOS, Windows) as well as mobile (Android, iOS) and web-based (HTML5) platforms.
>
> Godot is completely free and open source under the permissive MIT license. No strings attached, no royalties, nothing. Users' games are theirs, down to the last line of engine code. Godot's development is fully independent and community-driven, empowering users to help shape their engine to match their expectations. It is supported by the Software Freedom Conservancy not-for-profit.

Two main components of development in Godot are the nodes and the script that handles the functionality of it all. These two are explained in the following subsections:

3.1.1 Node System

Godot uses a node system as the fundamental infrastructure for the development of the games. A node can be an instance of an object, a property or a function. Nodes can also contain other nodes in them, which are called children, and the node containing the children is called a parent node. The children can have their own children further down the line. Similarly, a parent can be added to another node as a child to that node. The resulting structure is similar to that of the basic tree data structures used in C++ and Java. These nodes are used for all the scene building elements of the game development. So for example, let's say we want to place a tile set for a platform in a 2-D game. That tile set will be a node, that'll be a child of the main "world" node. The playable character itself will also be a node that'll also be a child of the main parent node. Hence, these nodes handle the look and physical features for every element in a scene.

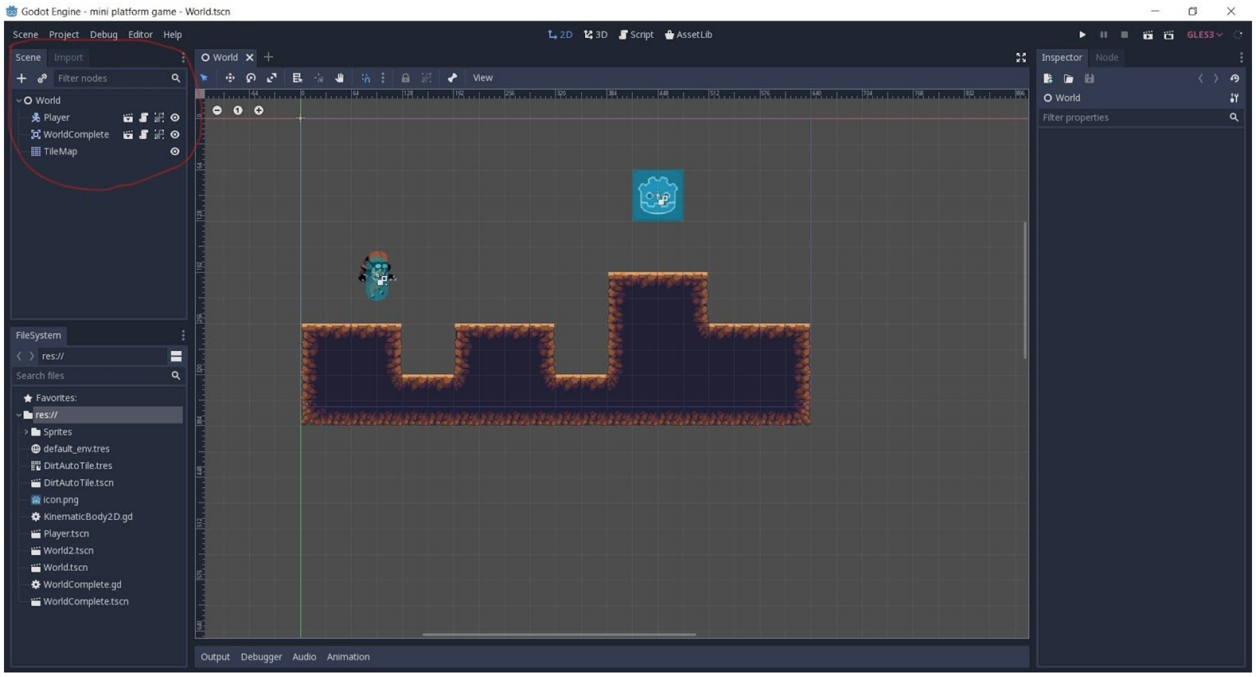
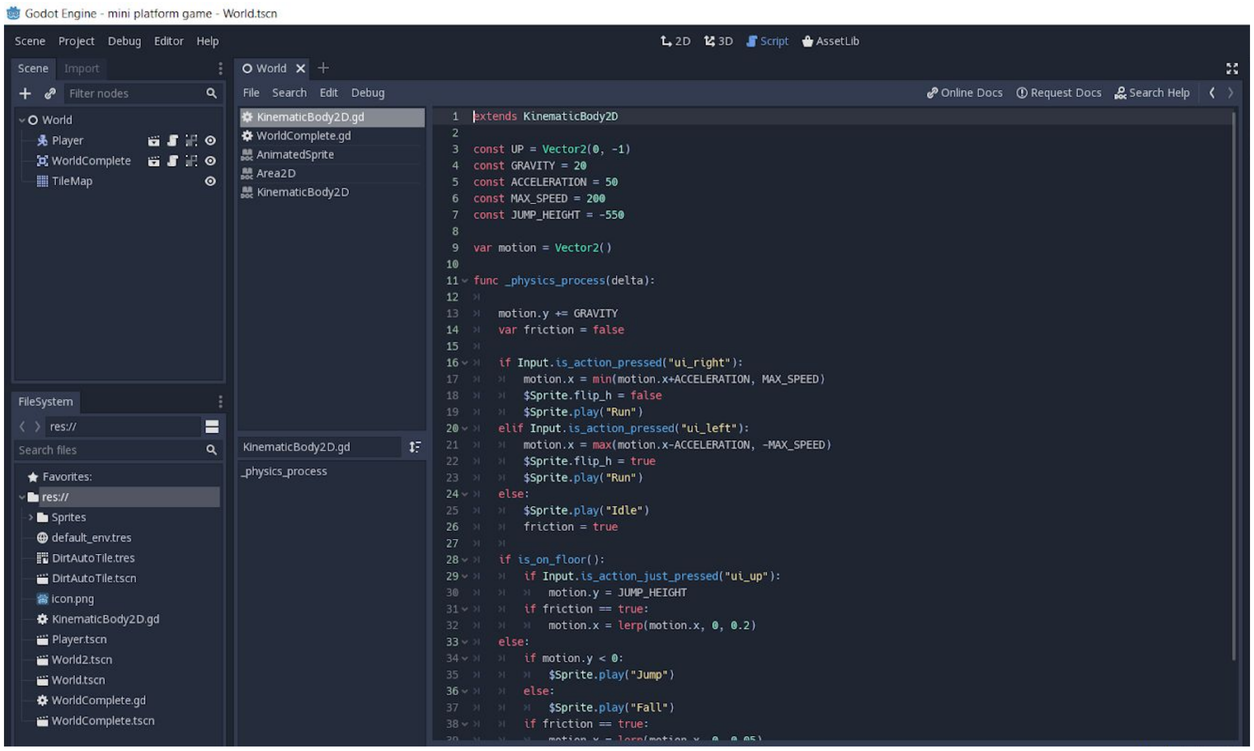


Figure 4: Nodes highlighted for a given scene

Figure 4 is a screenshot of the godot editor engine. At the top left of the image, the nodes present in the given scene are shown. The main parent node is the *World* node. It stands for the whole scene itself. Its children involve *Player*, *WorldComplete* and *TileMap*. *Player* is the playable character. *WorldComplete* is the blue box that can be seen in the editor window. Its function is to teleport the player to the next level once the character hits that blue box. The *TileMap* node handles the platform that the *Player* will be running on.

3.1.2 GDScript

In order to dictate the characteristics and behaviors of these said elements of a scene, Godot uses its own scripting language called GDScript [4]. It is a dynamic programming language that uses a syntax similar to that of Python.



```
extends KinematicBody2D

const UP = Vector2(0, -1)
const GRAVITY = 20
const ACCELERATION = 50
const MAX_SPEED = 200
const JUMP_HEIGHT = -550

var motion = Vector2()

func _physics_process(delta):

    motion.y += GRAVITY
    var friction = false

    if Input.is_action_pressed("ui_right"):
        motion.x = min(motion.x+ACCELERATION, MAX_SPEED)
        $Sprite.flip_h = false
        $Sprite.play("Run")
    elif Input.is_action_pressed("ui_left"):
        motion.x = max(motion.x-ACCELERATION, -MAX_SPEED)
        $Sprite.flip_h = true
        $Sprite.play("Run")
    else:
        $Sprite.play("Idle")
        friction = true

    if is_on_floor():
        if Input.is_action_just_pressed("ui_up"):
            motion.y = JUMP_HEIGHT
        if friction == true:
            motion.x = lerp(motion.x, 0, 0.2)
    else:
        if motion.y < 0:
            $Sprite.play("Jump")
        else:
            $Sprite.play("Fall")
        if friction == true:
            motion.x = lerp(motion.x, 0, 0.05)
```

Figure 5: Script preview for the player

Figure 5 shows the script that is connected to the *Player* node. This script controls the player movement that corresponds to different keyboard inputs. The constant variables are generally spelled with all upper case letters, while regular variables are all lower case. GDScript comes with several libraries that are well equipped to handle Godot specific development needs like working with 2-D and 3-D physics and other such mechanisms.

3.2 Multiplayer

Let's discuss the multiplayer aspects of the game. Within the game we need features that the user can interact with that do not interact with the world itself. Such features relate to networking and databases that go on behind the scenes of what a typical user would see. These 2 things are essential to make a multiplayer game experience smooth for the users. We wanted to focus on making a stable browser network that runs fast and delivers on low ping, and have multiple databases to store important information. This important information includes storing chat messages to active users, saving player names and sign in information, and item/build movement throughout the world.

3.2.1 Databases

The databases must contain any activity logged throughout the world to make the game work. That is due to the game being multiplayer over a web browser. If the game was just a solo player then we might be able to get away without using a chat and the player information database. In solo player games these things do not need to be logged, because there is no chat, no sign in information, and even though one user interacts with every item, these items and blocks are changed within the world itself and needs to be logged. Say there is a second player then we would definitely need to store all three of these things for a variety of reasons. In multiplayer, if another user interacts with the world and the database is not alerted of this then nothing will change for the other users view. With this database and everything runs smoothly from both perspectives. Multiplayer games also require sign in information that way you can recover from where you left off playing the game last time. Lastly the chat database is required to store all messages sent between players and display them to the active users. This database is important, because we want to store the messages, but only display them to users currently playing and not idle players. For this to be achievable we need to get information between the chat and player information databases to make this work.

Our databases will be created within MySQL, which is a leading database software free to access. As explained before we will have our 3 main databases, which is relatively simple to connect with the software. In MySQL you are allowed to connect as many databases as you would please, with all running on one host. We can then link tables to one another, which allows information to be stored in the two databases. This is the exact case I was talking about with chatting, but we need to know what player ID typed the message in the database. What I will need to make sure is that player ID's are linked to the chat table, so the correct name displays whenever a message is created. We will then store this message in the chat database along with the player ID. Many other connections will be needed as well, such as any link between items and players. The player will interact with items/the world more than anything else in this game without a doubt, so we need an efficient link between the two. We need to store every item type within the database, and link it to the player whenever a user collects something. For example, if user 1 went outside and mined a grass block we need to search for the grass block in the item database, and add one to the players inventory. We then need to update the game and account for the block that was lost within the world. Lasly user 1 goes into his house and puts the grass block in there instead. We then again need to update the player inventory and the world.

3.2.2 Server

When creating an efficient server one of the biggest things we worry about is game quality. We want to be able to host a game that provides very low ping and has no packet loss, but it proves difficult to do this. Most servers will have to make the choice of ping or packet loss (UDP vs TCP) [5], because these premade functions usually only allow one to work more efficiently than the other. In our game's case we chose low ping over packet loss primarily due to the fact that a 2D game is much less likely to drop packets than a 3D game. Given that this is a student project, we will not create a game capable of dropping many packets in the first place, so the choice was easier than it seemed it might be. The players ping is also heavily affected by the distance from user to the server [6], which in TUSG's case is not a problem.

TUSG will run its server through a router to provide an optimal experience for the players. This means the game will be localized as opposed to worldwide. The server itself will be run through the router that will contain a specific static IP that users will be connected to. Once the users connection passes through the router on this IP they will be allowed into the games title screen where they can sign in. This process is safer than an open server, because anybody wanting to cause any harm to the game server will have to be connected to the router. No outside sources can connect to the game so the safety of our world state is fairly high. The router will also have a designated port number that the server must be connected to.

Another thing our server will be able to handle is a server overload. Server overload is a difficult issue to deal with, because any user can be a troll and attempt to flood the game's databases. Client load balancing is a way to combat this, by having users load the game by access points [7], as opposed to the whole world as a whole. This seems like a lot of work without that much of a return since players would be near each other in the instance of TUSG. This problem is combatted simply by not allowing a large capacity in the backpack. TUSG does not have a large player base, so even if the number of players exceeds about 10 who are all mass dropping items there should not be much overload. This is also helped by the fact that standing near an item for about a couple of seconds will pick that item back up to reduce server lag. All in all TUSG has different actions in place to counter potential server overload.

4 Implementation Plan and Timeline

The timeline for the project will be divided into six checkpoints, each roughly 2 weeks long.

Checkpoint 1

We have already worked on the player movement and platform layout using the text editor Visual Studio Code [8]. So, setting that up for the game would be the first thing we do. This will include the two different setups for the player to be able to jump between. The whole environment will be available to run around in but the player will not yet be able to interact with any of it (outside of jumping over things wherever possible). At this point, the player will not be able to interact with anything in the environment.

The user interface will also have been put in play and for the most part will be interactive. This means that the player will be able to select their tool or put it away. They will also be able to look into their backpack. They will not have any materials available yet because they can't interact with any of the environmental elements.

The user interface will also include the chat feature. When the player presses *enter*, the chat will pop up at the bottom left corner of the screen. The player will be able to type into the chat box and it will appear on the "server" but since the multiplayer aspect is not set up yet, it will only be sent for the same user to see.

Checkpoint 2

For this checkpoint, we will be adding the sandbox mechanisms to the game. That will include mainly the mining and building features that were explained in the previous sections. After this checkpoint, the player will also be able to use the tool that they're provided with as it goes with the mining feature.

Speaking of mining, we will also iron out the rough edges that will be left out of implementation for the how the materials will work; both for mining and building. The different categories of materials will be given their own specific characteristics. The player will then be able to obtain those materials from their respective sources and will be able to use those materials in their intended manner.

Checkpoint 3

We will have a proper single-player game running by this point, with all the gameplay features in place. The next step from there would be to export this game in javascript and run it on a web browser. Going along with that, a server will also be up and running, in order to handle single players running the game on their own web browsers.

In addition to this we will now need to have our 3 main databases storing item information, user information, and chat information for an efficient multiplayer experience. As explained in section 3.2, these databases rely on one another and will need to be linked together through MySQL. Now that we have a user information database we will need to have a sign in/up system running, where this information can be stored within the user database. Running this without the server finished may cause problems that will be addressed in checkpoints 4 and 5.

Checkpoint 4 & 5

Up to this point through the timeline we already had an extremely basic network running [9]. We want to expand upon this system in the following ways:

A server is no simple task, and we want an entire extra checkpoint to make sure the server is smoothly operating and capable handling multiple users at once. This would include having a static IP address for a router, that all the players will play through. This allows for our server to be much more manageable than one that runs forever, and broadcasts throughout the country. This simple server routing should allow for minimum packet loss as well as consistent ping.

When the server has been completely finished, the user account information being stored will become usable and stable. Information will not be "lost" and will always be stored and ready to be accessed. Completing the account information before the server may cause problems, and we may not know if the system is working 100% correct, so we will shift some focus in these two checkpoints to make sure that user information is not dropped and runs efficiently.

Checkpoint 6

The main goal at the end of the semester is to have our poster board finalized and ready to present. We will go through every little detail that we put into the game and select the most important highlights that differentiate TUSG from other sandbox shooter games. This should include the most significant ideas behind the creation process of the game.

The Workshop is an add-on that we would like to put onto the game, but could provide significant problems to the game that should be quite polished at this point. The idea behind the workshop is to give users an open source to create anything they want within the world. This workshop is just an experiment within the checkpoint to see how this could possibly work in our currently running version of the game.

5 Conclusion

Today, video games seem to be the same copy and paste trap, and everyone continues to fall into the cycle of capitalistic greed. Whether it is shooters, open worlds with limited gameplay mechanics, or sandboxes with one objective, no game seems to break through the barrier that gaming companies have created.

TUSG proves that it can burst through this bubble and allow for more seemingly endless/new gameplay. TUSG takes the three main game genres stated above, and combines them into one wholesome experience. This game can be any gamers cup of tea, weather they like fighting, relaxing, or adventure. Hardcore gamers can perform all of the above and really push creativity to its boundaries.

All the players will be given the ability to unleash their creativity into this world, weather it be for fighting or building a masterpiece. Combining these two mechanics allows for new experiences in the realm of 2D shooters. The capability of building specific fortress' or bunkers while in action is something we have not seen before, other than the smash hit *Fortnite* in 2017. In the 2D spectrum it is new and has a different feel as you have a much better feel for what your opponents next move will be. Combine this suspenseful engagement with the peaceful building setting and we have two completely different games in one. The game can be played however the user intends, making the experience a remarkable one.

In order for a user to access the game, the server is being ran through a router that the players must be connected to. This connection will check if users are harmful when entering the server, and if it decides they are clean, the player is allowed into the server. Once a user is allowed into the server then if will begin recording their activity, in other words they can receive chats and interact with the world.

The player base can have a better gaming experience by chatting to their friends within the server. This communication between the client and server showcases exactly how efficient the server will be. Messages will travel between all players active, and will not be sent to players that are inactive. All of this data will be observed within the databases, and the appropriate information will be sent back from the server to the user. This basic server will be accountable for respectable ping as well as minimal packet loss in the game.

6 Bibliography and work cited

[1] Steam Support. *Configuring a Router for use with a Dedicated Server*, 2017, https://support.steampowered.com/kb_article.php?ref=5121-RPXB-7955.

[2] Gouirhate, Noufel. *Build a simple chat app with node.js and socket.io*, Dec. 24, 2017, https://medium.com/@noufel.gouirhate/build-a-simple-chat-app-with-node-js-and-socket-io-ea716c093088.

[3] Godot. *Introduction*, 2019, https://docs.godotengine.org/en/3.1/about/introduction.html.

[4] Godot. *GDScripts basics*, 2019, https://docs.godotengine.org/en/3.1/getting_started/scripting/gdscript/gdscript_basics.html.

[5] Godot. *Networking*, 2019, https://docs.godotengine.org/en/3.1/tutorials/networking/index.html.

[6] Chris “Battle(non)sense. *How netcode Works, and what makes ‘good’ netcode*, Oct. 25, 2017, https://www.pcgamer.com/uk/netcode-explained/.

[7] Ruckus Wireless. *Client Load Balancing*, https://docs.ruckuswireless.com/unleashed/200.1.9.12/c-LoadBalancing.html.

[8] Visual Studio Code. *Getting Started*, 2019, https://code.visualstudio.com/docs/?dv=win.

[9] Gamefromscratch. *Networking -- Godot 3 Tutorial Series*, Sept. 11, 2018, https://www.youtube.com/watch?v=JuRhRhJ2914.