Game Design Document

A game design document is a living document that describes the intent of the game design. It has two goals. First, it documents the decisions that have been made about the game and communicates those concepts to the entire team. Thus, it needs to be detailed enough for programmers to refer to when they need clarification about any game aspect. It must be able to be updated as the game is to be built. The need to have a game design document increases with the size of the team and the length of the project.

For a student project, the intent is to capture as much as possible of your design. The game design will be larger than what you can achieve in a semester, but you must decide what you need to do first. This document should be in version control so that you can see it changing and growing. Given that we are using git, you could also use @name to assign design parts to individual team members.

Overview

{some_name}

Mikkel Aas, Ruben Christoffer Hegland-Antonsen, Mikael Falkenberg Krog, Erlend Johan Vannebo, Michael-Angelo Karpowicz

Game Concept

You play as a druid that travels different biomes/areas to fight different factions of animals. Once you defeat a boss (an animal), you claim its soul granting you the ability to transform into that animal.

Genre

2D Platformer

- Hollow knight, Ori and the Blind forest

Target Audience

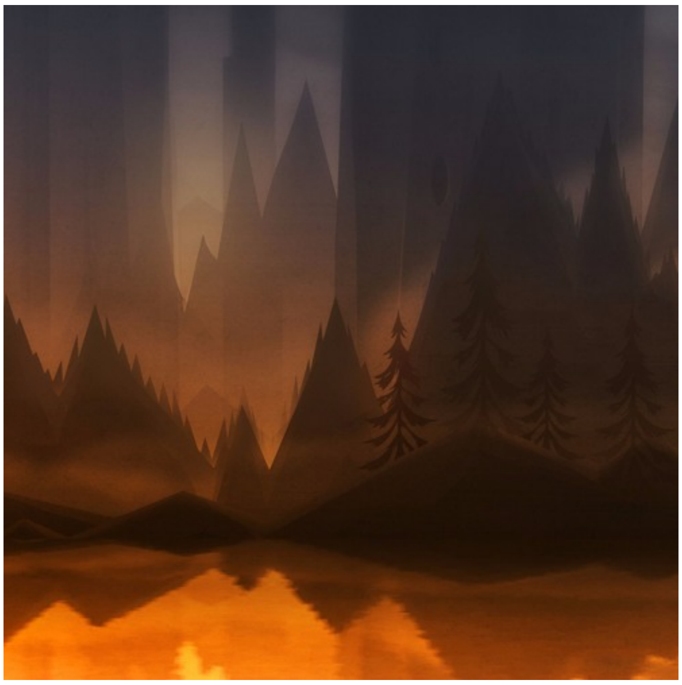
Teens/Young adults

Game Flow Summary

Linearly, from left, right, up, down, on a platform. It will be a semi-open world—freedom to go where you please (to some extent).

Look and Feel

https://www.youtube.com/watch?v=wOJAErCvVhs



The game will have a nature-based theme; this works well with the druid and animal transformations—different biomes like forests, snowy areas, desert areas, etc.

Gameplay and Mechanics

The player traverses through a semi-open world where they are met with obstacles, puzzles, and enemies. Each scene or biome will contain one boss you must defeat to continue to the next level/scene. By defeating the boss, you gain its attributes that will increase the complexity of the combat and puzzles by adding new abilities to your character.

Gameplay

The player will use a set of animal attributes and abilities to make their way through the game.

Game progression

Very clear progression system with different bosses, unlocking new abilities along the way. Some sort of leveling system with different base stats.

Mission/challenge Structure

In the beginning, the player will have a very limited set of abilities. The combat and the puzzles will therefore be straightforward to complete. When you gain more abilities by defeating the bosses, the challenges will become increasingly more complex.

Puzzle Structure

There will be some puzzles, mostly small ones, that can be solved using different animal transformations.

Objectives

The player's main objective is to defeat all the bosses.

Mechanics

What are the rules to the game, both implicit and explicit?
This is the model of the universe that the game works under.
Think of it as a simulation of a world. How do all the pieces interact?

- Entities have health
- Break objects
- Combat
    - Real time
    - Not turn-based
    - Melee and ranged
    - Good positioning and good timing required

- Movement
    - Jump
    - Double jump
    - Sprint, Dash
    - Fast fall (enhanced gravity)
        - Not all animal shapes will be able to do every movement
        - Fall down quickly as a bear, take fall as a cat (puzzle?), glide as a bird (fly on the wings of love), etc.
- Transformations
    - Owl
        - Glide
    - Bear
    -

Physics

Physics will not be favored over fluent movement. We will have some physics, such as collision with other entities and gravity. Friction, gravity.

Movement

The player will move / attack/interact by using a keyboard. Though, the mouse can be used to navigate the interface/UI. The player will also have the option to play with a controller.

Objects

- Platforms! (some moving)
- Obstacles
    - Objects that needs to be destroyed or moved to progress
- Enemies
    - Smaller entities that needs to be dodged/kill to prevent yourself from doom
- Bosses
    - Big entities that when slain will allow you to enter the next scene/biome
- Soul
    - currency
- Blood
    - health

Actions

What are the other interactions the player has with the game world?

- Runic checkpoints
- Hidden walls
- Picking herbs / flowers / mushrooms in small plastic bags

Combat

If there is combat or conflict, how is this specifically modeled?

- Druid
    - Melee stick (kjepp)
- Bear
    - Higher defence
    - Heavy hitting, slow attacks
    - Heavy (falls faster, moves slower, jumps low)
- Cat
    - Agile
    - Low damage, quick attacks
    - Lightweight (absorbs fall damage, faster, jumps high)
    - Night vision (?)
- Owl
    - Can semi-fly (hover)
    - Ranged attack (?) (ohoooooooo)
    - Fear? (**o**hooo**oo**oooouho**oo**ooooo >:))

Economy

Killing enemies grants you...

Animal soul points.

Soul points can be used to upgrade animal specific stats. For each animal upgrade the druid receives an upgrade (health etc.)

Scene Flow

Each scene represents the biome of the animal-boss. When the boss is defeated there is a small travel sequence before the next biome is loaded.

Game Options

Difficulty modes. Adds multipliers to mob/player health, damage upgrades etc.

Replay and Saving

- Borderlands-style replay, second time is harder, but you keep upgrades
    - In theory, endless upgrades
- Save after each checkpoint
    - Save inventory, levels, etc. anytime (automatically)
    - Respawn at checkpoint

Cheats and Easter Eggs

This part will not be prioritized

- No cheats.
- Unlock rabbit as easter egg, all it does is lay easter eggs
    - 1hp, challenge to complete the game as the rabbit

The Story, Setting, and Character

Story and Narrative

The world has been corrupted, and you, as a druid, must save nature and the souls of the animals. The game will not have a big focus on the story, but some small story elements will be present. Some minor cut scenes can be included, for example when changing the scene or meeting the boss.

Game World

The world will be in nature, with nature being corrupted and dark when entering a new area. When the boss of the area has been defeated, the area will become lighter.

General look and feel of the World

Mostly dark and gloomy nature look.

Areas

Each area will be based on a different biome that corresponds to the natural habitat of the given animal boss. Forests, deserts, icy area etc.

Characters

The game will not include a lot of characters, except the main character.

Levels

Playing Levels

Each level should include a synopsis, the required introductory material (and how it is provided), the objectives, and the details of what happens in the level.
Depending on the game, this may include the physical description of the map, the critical path that the player needs to take, and what encounters are important or incidental.

Forest:

- Owl boss
- Gains the owl transformation after defeating the boss

Desert:

- Cat boss(sphinx)
- Gains the cat transformation after defeating the boss

Icy:

- Polar bear boss
- Gains the polar bear transformation after defeating the boss

Final boss area, final boss tba. Maybe a god / demon?

Training level

How is onboarding managed?

At first the player will learn how to move and attack. Upon acquiring a new animal spirit, the payer will receive a quick tip on how to utilize it.

Interface

Visual System

If you have a HUD, what is on it? What menus are you displaying? What is the camera model?

- Health bar
- Currency
    - Displays how many animal souls you have collected (they are corrupted so it's fine)
- Amount of herbs / shrooms
- Current animal
    - Display the current animal in an icon

Control System

How does the game player control the game? What are the specific commands?

Possible option for binding your own keys

- Default keyboard
    - WASD: move
    - Space: jump
    - j: Primary attack
    - bnm: Transformation
    - 1: Smoke herb
    - e: Interact
    - shift: sprint
    - dd: dash right
    - aa: dash left
        - This has to be tuned so that you don't accidentally dash when you just want to make several small movements.

Audio, Music, Sound Effects

https://www.youtube.com/watch?v=vdwYUbVtR6U

Help System

Artificial Intelligence

Opponent and Enemy AI

The active opponent plays against the player and therefore requires strategic decision-making. The boss changes state during combat. Would be appropriate to use a finite state machine here. Each state is designed to be in favor of certain animals.

Non-combat and Friendly Characters

-

Support AI

- Liten hagenisse

Player and Collision Detection, Path-finding.

- Pathfinding
    - Enemies walk back and forth on the platforms. When the player is nearby, they will walk towards the player.
    - Will collide with enemies

Technical

Target Hardware

- Anything
- It is not hardware cost heavy

Development Hardware and Software (including the game engine)

- Unity 2020
- Linux, Windows

Network requirements

- Dial-up connection
- Essentially none
- Mostly single player
- Network is required for the multiplayer gamemode

Game Art

Key assets

How are they being developed? Intended style.

The art will be developed in photoshop. The style will be pixel art.

Something like this:

Ideally something like this:

NTNU

Norwegian University of Science and Technology

Platformer Group Discussion

December, 2021

Mikkel Aas
Ruben Christoffer Hegland-Antonsen
Michael Angelo Karpowicz
Mikael Falkenberg Krog
Erlend Johan Ringen Vannebo

Strengths and weaknesses of Unity

Strengths

Good tutorials, documentation, and a big community

Unity is one of the most used game engines and has a huge community with over a million active users each month. This means there is a high demand for tutorials and guides on how to do many different things in Unity. Therefore, there are plenty of good resources such as video tutorials and forum threads on the internet to help you learn and understand Unity. Unity is also very thoroughly documented, with great descriptions and examples of everything Unity offers. We found these aspects of having Unity as our game engine very useful because we could look up anything we were unsure of.

C# as main scripting language

C# is a very high-level, flexible programming language which makes it a lot easier to learn than lower level languages such as C or C++. C# is similar to Java in many ways and we have used Java before so a lot of the syntax and how C# work was familiar to us, and that helped us a lot during the process of learning the language. C# is also well documented which was a huge positive. We also didn't encounter any C# specific problems, so we were very happy with the process of learning and using C# in our Unity project.

Built-in physics engine

Unity uses PhysX for 3D and the Box2D engine for 2D, which simplifies physics calculation for your game. It's also very fast and relatively easy to use. We used the physics engine a lot for creating character movement in our game. In the beginning, we had some issues where we did not use the physics engine properly, by for instance moving the Transform of an object directly instead of using the Rigidbody, but we figured out how to properly do it in the end.

Mature Game Engine

Unity has been actively developing the engine since 2005 and has matured over the years. This means that they have seen a lot of improvements and provide good support. It is also well known and could be seen as an industry standard along with the Unreal Engine.

Build once, reach billions

Write once, build to any platform. Unity supports a wide range of platforms to deploy to desktop, mobile, and consoles. We decided to build for the web as it is the easiest to share and requires the least UI settings in-game as the web provides it.

Big asset store

Unity's asset store contains every asset an indie developer needs. It has visual scripting tools, so you do not even have to know how to code to make scripts for your game. But most importantly, it has a wide variety of art for every style, which means that we did not have to spend too much time making our own sprites and animations. However, we did create the sprites for our levels and collectibles.

Prefabs

Prefabs are easy to make and use and allow you to create templates of your game objects. This allows you to easily place multiple enemies for instance, and can also be used for instantiation. They do not always work as well with version control though if multiple people change a given prefab.

Serialized fields

Serialized fields make fields accessible from the Unity editor. This is very helpful when you, for example, want to test the effect of different field values and get instant feedback from the game. Furthermore, it helps analyze the change of field values as you play the game. In our project, we used it, for example, to find the perfect speed for our main character and their enemies. We also used it to see that the player's health field was accurately updated when the player took damage.

Weaknesses

Poor built-in team collaboration tool (Unity Collaborate)

Unity Collaborate is a tool within Unity that you can use to collaborate in a team. Although Unity Collaborate tends to make it easier to share changes in the same scenes, it is severely limited by lack of features compared to git.

Difficult to use Unity with git

Version controlling became quite difficult when it came to merging prefabs and scenes. The latter was the most difficult, as it was practically impossible to merge two scenes without breaking the game. The files were also practically unreadable during code review, so it was difficult to depict what specific changes were made to the scene. Instead of dealing with the merge conflicts within git, we created copies of prefabs and scenes and worked on the copies instead. These copies were later combined manually within Unity in a meeting.

Closed source

Most of the source code for Unity is not disclosed unless you pay for the commercial source code license. They have released the C# portion of the engine and editor as reference only which can help gain an understanding of how you can use some of the functionality, but it’s not possible to modify or build on the existing code without the commercial license.

2D Tilemap

The Tilemap system in Unity is used for easily placing sprites in a tile-based map, where each tile is a fixed size, which was 16x16 pixel size we primarily utilized. Although this can be convenient for placing a lot of sprites at the same time, the Unity Tilemap is known for having problems. One of the problems we encountered was aliasing. By default, Unity will not render the tile map pixel perfect and it can cause some render issues. In order to fix this issue we had to disable or nullify any compression, set filter mode on Point, disable any anti-aliasing on our project, and envelop our sprites into a sprite atlas to make it even clearer to Unity that it should not try and alter the rendering of our pixelated images.

Process and communication systems during development

Work Process

During the development process we had two regular meetings each week. These were mainly for showing each other what we had done and assigning new tasks to each team member. In one of the meetings we normally merged new features and had code peer reviews, and in the other meeting we planned and assigned new tasks and started working on them. During the rest of the week we would sometimes do smaller impromptu meetings if we felt the need to collaborate on something. Otherwise the group members would work independently or in smaller groups of two or three for most of the week.

This work process worked for us, and allowed each team member to decide themselves when they wanted to work. This was smart because most of our team members have different schedules and arranging more meetings would be quite challenging.

During the development process we utilized the mvp playtesting to improve our game. Although, all of the feedback from the MVP playtesting were problems we already knew, so this did not help us a lot. However, we did get some positive feedback as well. This way we knew what we had done well.

Roles

We did not have any designated roles, but people often ended up working within certain fields. For instance, Michael made all the map sprites, map layouts, and maps in unity. While the rest of the group did work where it was needed. Since we wanted all the group members to have as much learning outcome as possible, we did not assign specific roles to people because this would reduce each person's overall learning outcome within the field of game programming.

Communication systems and file organizing

For arranging and holding the meetings we used Discord. This was the easiest considering we already had a discord server for the course and our own text and voice channels.

We used Google Drive for file sharing of all our documents, because it made organizing easy. We would use Google Drive for meeting notes, the game design document and game level sketches.

Coding style

A general mark of code quality is consistency, which leads to predictability. Code consistency includes standard naming conventions, consistent indenting and spacing, and standardized architecture.

In our code, we agreed that all names should follow camel casing, classes and functions should begin with a capital letter, and private fields would start with an underscore. We also decided that the opening curly brackets should be on the same line as a method or conditional statement. Having opening curly brackets start on the same line made it possible to see more code at once and save vertical space.

We agreed on a standard architecture where constant fields were declared at the very top of the class. Right after came field declarations, then methods. Within these groups, the order of access ranged from least accessible to most. However, we had no public fields to ensure efficient encapsulation. We used the style cop GitHub repository as a reference when deciding on architecture.[1]

Ideas that were not implemented

During the game development process there were many ideas floating around that we did not implement. Most of these were not implemented due to time constraints.

[1] https://github.com/DotNetAnalyzers/StyleCopAnalyzers/blob/master/documentation/SA1201.md

Multiplayer gamemode

At the start we were discussing having a multiplayer mode where you could fight other players. We also discussed having the entire game playable in multiplayer mode. We decided to not do any of these and focus on the single player aspect.

Speedrun-Parkour gamemode

We discussed making an optional parkour/speedrun map where there were no enemies and you tried to get through a very difficult map where you had to use all the movement abilities in the game. This was not prioritized to focus on more important matters.

More Hidden secret rooms

The hidden rooms were originally intended to not be visible before you had entered them, which would have made it a bit harder to find them. We ended up not doing this because we did not think this was a high priority issue, and instead chose to work on more important issues.

Player transformation complexity

The combat system was supposed to be more complex. We wished to develop a combat system where each transformation had at least one special attack that you could use in sequence with the other transformation-dependent special attacks to make for a more unique and dynamic playstyle. Without prior experience in game development, we quickly realized that this was out of reach and had to be down-prioritized to make sure that we reached an MVP.

The transformation states were also supposed to have at least one set of unique movement abilities. The owl transformation, for example, was supposed to have a gliding ability instead of being able to jump more than the other transformations. Unique movement abilities were not prioritized to ensure that we reached an MVP, which resulted in the fact that we never had time to develop it.

Puzzle elements

There were supposed to be puzzle elements in the final product, but this was also not prioritized due to time constraints. The original idea was to have puzzles where the player had to use specific animal transformations to solve them. Because of the animal transformations not being fully implemented as planned we decided to not make any puzzle elements.

Sprites and animations

Unity's asset store contains a lot of art, but unfortunately, not everything is free. We could not find any free sprites for our 2D pixelated polar bear boss nor our 2D pixelated desert cat boss. Neither could we find animations and sprites for each player transformation and the transforming animations between each transformation. This meant that we had to improvise.

It turns out that creating a sprite for every animation state is a very time-consuming job that would halter the progress of more prioritized aspects of our game. Therefore these sprites and animations look very underdeveloped in our final game. The different player transformations, for example, just change the color of the player sprite. And the bosses do not have any animations.

Transformation unlocking after each boss

Player transformations were supposed to be unlockables that you would be granted after killing a boss. We never finished all the levels, so keeping the transformations locked would serve no purpose as you would never be able to unlock all of them. Therefore the player can use all transformations from the beginning. Though utilizing the owl transformation with seemingly unlimited jumps is a bit overpowered.

Use of version control systems, ticket tracking, branching, version control

Git

We used git for version control. In the beginning, we tried out the Unity Collaboration tool, but decided that it was easier and more reliable to use Git (and only Git) for version control. In hindsight, we believe this was the right choice because we knew Git from before, and Unity Collaborate would not be able to offer us something other than easier scene merging. Although this would have been useful, it does not add up for all the extra features you get by using Git.

Issues

In the beginning we used a planning document on Google Docs instead of Git issues, but later changed this to track all our progress using github issues because it was nice to have this in our repository. It is appropriately more organized in github and allows us to easily assign or label issues to be concluded. It also helped us to easily receive feedback from students who have played our game and notice problems or issues that we could have overseen.

Workflow and Branching

We tried to utilize Git Flow as our workflow[2], meaning we have a *master* branch that is supposed to be stable, a *dev* branch for development (unstable), and feature branches for each feature. When finished with a feature, we opened a pull request on Github and had to get it reviewed by at least one other member before being allowed to merge it into the development branch. We used rebasing instead of merging when combining two branches.
In order to merge from the development branch to the master branch, four other group members would have to approve it and should only be done for major changes (releases). The MVP and finished game are examples of major releases.

[2] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

We utilized hotfix branches which fork directly off the master branch and are merged with both the master and dev branch. This allows fixes to be made to the master branch without merging from dev which is reserved for releases.

Final thoughts

While the final product may not be exactly as envisioned, and some ideas were not implemented, we feel that the final product is of a quality that is expected considering the time frame. The group has had good teamwork throughout the entire project, and we've had no problems in the group. We have learned a lot about game development. From game design concepts about what psychologically makes a game fun, to the technical side of coding and using Unity, it has been a very educational process.

Game pitch

Genre: 2D platformer, adventure, like Ori and the Blind Forest and Hollow Knight

Basic idea: You play as a druid that travels different biomes/areas to fight different factions of animals. Once you defeat a boss (which is an animal) you claim its soul granting you the ability to transform into that animal.

Animal: I was thinking that you could use the different properties of the animals to travel the world and solve small puzzles and use them in combat. For instance, a bear is heavy, strong, and beefy, but slow while a cat is agile and stealthy, but very fragile(no fall damage).

Combat style: I was thinking the enemy fighting style is more like dark souls, where you must recognize the enemy's attack patterns and use the different animal forms strategically to be able to defeat them. I was also thinking that defeating normal enemies you gain some sort of points that you can spend upgrading your animal, for instance making you bear either stronger (more damage) or more beefy (more health) that way you can create your own way to combine and use the different animal-forms as well (also adds for replayability with different builds)

Some story ideas (not finalized at all, just some ideas, and the story shouldn't be too much in focus) could be that the shaman is sort of like Noah's Ark, where he must collect all the animal souls before the world collapses. Or the final boss is god, you need to use what he has created and defeat him, you become a god and create a new and better world for these animals.

Engine: Unity

Challenges: Time! Graphics, animation, and sound are really important in these types of games.

Notes

Make sure MVP is fun - switching fighting
Splice for music perhaps

Planning

Meeting notes

10/9-2021

- Several smaller scenes
- Dark souls ish
- Stats
    - Improve abilities that's already there e.g. cat form moves fast, upgrading makes it move faster.
- Main focus on combat, not the puzzles and obstacles.
- Movement should be very fluent, don’t limit the player movement, but add map restrictions.
- Druid symbols
- The weather/environment changes upon defeating the bosses

13/9-2021

- Bruke runer (checkpointsa) for å snakke med spirit-godsa for å oppgradere dyre spirit egenskapene
- Ugle og katt kan se i mørket, som gjør at du må bruke de i ulike områder
- Spike floor / ceiling, owl puzzle
- Hidden wall puzzles

- Til neste møte
    - Få unity opp å kjøre (fix ide, unity, etc.)
    - Gjøre seg kjent med unity

17/9-2021

- To do:
    - Basic character
        - Prototype sprite
        - Player movement script
        - Basic physics (jump & collision)
        - Animal transformations
    - Basic level for testing
        - Platforms
        - Moving platforms
        - Background
    - Basic entities
        - Shared script for entities
            - Health
            - Damage system
            - Economy (Soul points)
    - Saving system
        - Checkpoints
        - Automatic saving (refer to DD(design document) for more details)
    - UI
        - HUD
            - Refer to DD
- Til neste gang
    - Unity Fundamentals
        - Gameobjects
        - Scenes
        - Basic scripting (C#)
    - Unity 2D development
    - Basic platformer

24/9-2021

- To complete before next meeting:
    - Hud
    - Player movement
        - Double jump
        - Wall jump
    - Level design
        - Tutorial stage
    - Enemy script

30/9-2021

- Enemy script fix
    - When using RigidBody2D, do not change the position of the entity with transform directly (very bad practice)
        - Change the script so that the entity uses the rigidbody to change its position.
- Merged player movement, enemy script, tutorial level, and UI/HUD
- Remember to make prefabs of entities
- Make meetings shorter by quickly assessing problems that needs fixing (not focusing on fixing problems during the meeting)

1/10-2021

- To next time:
    - Player movement (Ruben):
        - Dash
        - Gliding (slowing down gravity)
        - Fix double jump input (currently always jumps when space is pressed down)
        - Try to allow horizontal movement when wall-jumping (jumping *off* wall instead of climbing)
    - Enemy movement (Mikkel og Mikael):
        - Fix rigidbody
            - Useful: https://docs.unity3d.com/Manual/class-Rigidbody2D.html
            - Useful: https://docs.unity3d.com/ScriptReference/Rigidbody2D.html
        - Make different type of enemies
            - Flying
            - Jumping
            - Shooting
                - (f. Ex Stationary entities that shoot projectiles)
            - Hiding
                - Peekaboo
            - Melee attacking (simple forward slash)
    - Obstacles (Michael)
        - Spike (w/ wall spikes) (Thorns instead for nature theme?)
        - Fire/Lava floor
        - Moving platforms
        - Fake platforms?
        - Puzzle doors? (corrupted plant doors need some form of key(soul?))
        - Destructible objects?
    - Tutorial level design specifics
        1. Learn simple jump

2. Meet first enemy
3. Learn fake objects/destructibles
4. Find the first locked door.
5. Find first key
6. Find unmoving platform
7. Find lever for activating moving platform
8. End of level with double jump as power-up reward

- Abstract class for entities (Erlend / Ruben TBA)
    - Health
    - Common functions
        - Die()
        - Respawn()
            - Events for these
- Make Main Menu (Erlend)
    - Nice Brackeys video: https://www.youtube.com/watch?v=zc8ac_qUXQY
- Read through Unity documentation (Everyone)
    - Unity - Manual
    - Use arrows to go to next section

8/10-2021

****Priority by order****

- Abstract Entity (player, mobs, boss, etc.) class [1 pers] (Done)
    - Health (hearts)
    - TakeDamage(int damage)
        - Decrease health
        - Fix animations
        - Respawn (if applicable)
    - (abstract) Respawn()
- Combat system (melee) [2 pers]
    - Ideas
        - Raycasting
            - Send out ray and check if it hits enemy
        - Overlapsphere (probably the best)
            - Create an invisible circle in front of enemy with a given radius (attack radius) and checks if enemy collider is within it
            - If we hit an entity, call its TakeDamage()
    - PlayerEntity class (extends Entity)
        - Will check if input in Update and perform an attack
    - Boss [Ruben]
        - Polar bear
        - It's possible to use existing CharacterController2D for movement
        - FSM would work well for our purpose
            - Bear States:
                - Melee attack player (simply attack the player)
                - Charge attack
                    - Attacks everything in its path (invincible effect)
                - Shockwave (Particle systems)
                    - Jump and smash its fat belly on the ground
                - Tired state
                    - Vulnerable
    - Simple enemies
- Main Menu (UI)
    - Start and quit
- Attach player on moving platforms (player script) [Ruben]
    - Easy fix
- Transformations (Mikkel)
    - PlayerTransformation script
        - Enum Role (druid, bear, …)
        - Transform(Role role)
            - Will change CharacterController2D Properties depending on role
            - Change speed
            - Gravity scale

- Level (Michael)
- Puzzle elements
    - Lever/Plates/Key
- Sprites & Animation

15/10-2021

- More on transformations (Mikkel)
    - Allow for different animation, sprites, etc.
        - Easiest is probably to have all of these components in child gameobjects and disable / enable these gameobjects based on current transformation
- Boss entity (Ruben)
    - Attacks
        - Normal, charge and shockwave attacks should all damage player
    - Should be able to die
    - Invincibility effects
- Checkpoint system (purely logical / no sprites at this point) (Erlend)
    - Need a start position on each scene
    - Player respawn at checkpoint / start position
- Thorn damage to player (Mikkel)
    - Add a collider to each thorn
        - Make thorn prefab
- Connect HUD to combat / health system (Ruben)
- Basic Main Menu (Erlend)
- Sprites / Animation
    - Player appearance
- Entity (Mikael)
    - Knockback
    - Attack
- Player sticks to platform when on it (Ruben)
- Applying level sketch on game scene (Michael)

22/10-2021

- [x] ~~Fixes (Ruben)~~
    - [x] ~~Fix player respawn~~
        - Fix null references
    - [x] ~~Fix enemy respawn~~
    - [x] ~~Fix bugs (outlined on github)~~
    - [x] ~~Fix normal enemy damage~~
    - [x] ~~Die when falling outside map~~
    - [x] ~~Fix polar bear being smol~~
- [x] ~~Take damage when colliding with enemy (Ruben)~~
- [x] ~~Moving platforms on Test Level (Michael)~~
- [x] ~~Fix Level bits & Place enemies (Michael)~~
- [x] ~~Boss arena (Michael)~~
- [x] ~~Audio feedback (Erlend)~~
- [x] ~~Thorns damage (Mikkel)~~
- [x] ~~Entity damage red flash (Mikael)~~
- [x] ~~Maybe fix knockback if time (Mikael / Ring Ruben)~~
- [x] ~~Thursday: Put everything together~~

29/10-2021

- See github issues!

05/11-2021

Higest-lowest priority:

- [x] ~~Fix big bugs~~
    - [x] ~~Bouncing on platform issue~~
- [x] ~~Main Menu~~
    - [x] ~~Buttons work~~
    - [x] ~~Options menu with volume control~~
- [x] ~~Souls on kill~~
- [x] ~~Blood bar mechanics~~
- [ ]

- Available Start level stuff
    - Player
        - Double jump
        - Sprint
        - Attack
        - Slide
    - Enemies
        - Goblin

- Workflow
- Git workflow.

Networking

- No networking

Weekly meetings / work sessions

- Thursdays 10:15 - 12:00
    - Presenting what we have been able to do since the previous meeting
    - Peer-reviewing
        - Positive feedback and room for improvement
- Fridays 12:15 - 14:00
    - Assigning tasks for the next friday
    - Coding together over discord
        - Able to ask questions during the early stages of the task

People’s exams

- 3. December
- 9. December
- 14. December
- 16. December
-