




WINTER SEMESTER 2024/25

ALGORITHMIC CREATIVITY – PROCESS BOOK

ŠIMON ŠKVÁRA

UNIVERSITY OF EUROPE FOR APPLIED SCIENCES

Lecturer: Jörg Reisig



Prototyping

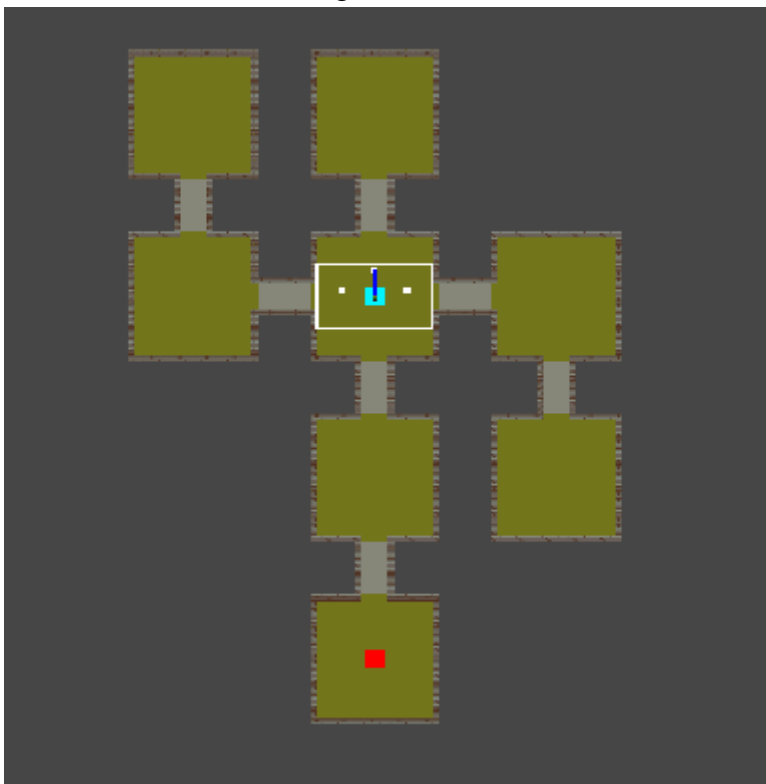
Prototype 1

Sort of like an endless runner, but instead it goes down. There is a wall on each side and the player can jump between those walls. The random topic here is the random obstacles and the obstacles will be enemies who spawn on the walls. The player can kill those enemies by jumping to the other wall. There are 3 enemies in this version. One that just hangs on a wall and shoots projectiles at you, one that is running on some surface back and forth between the walls and an enemy that homes in at you and you have to jump to kill it.

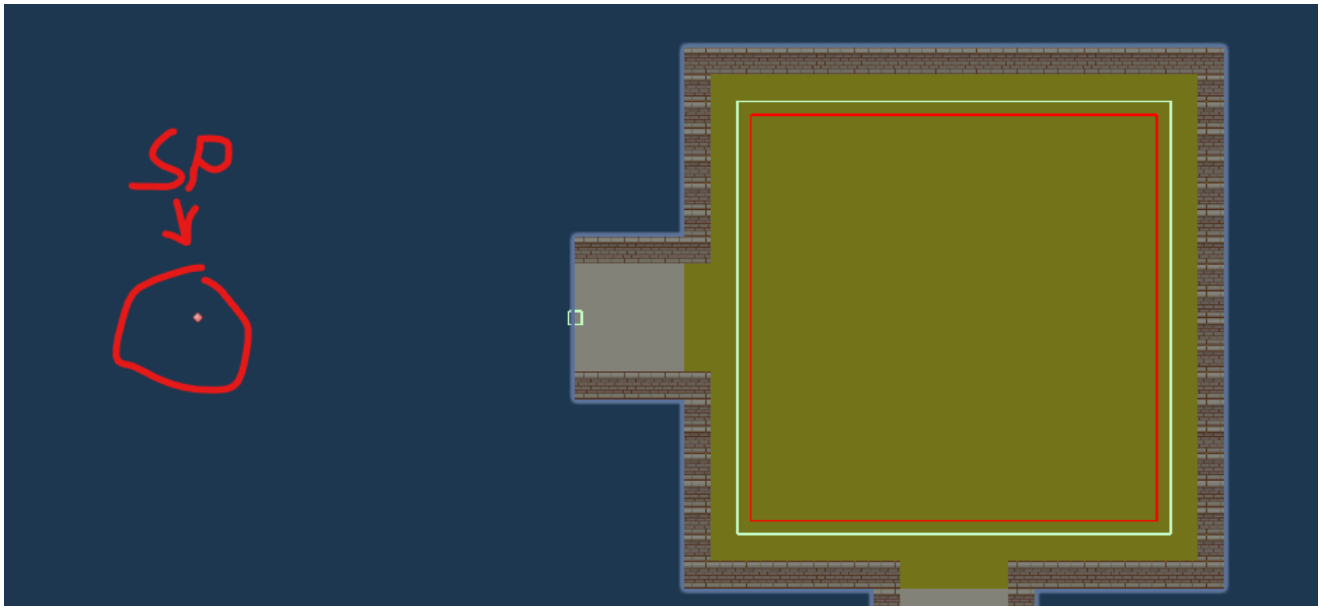
Week 2 (8.10. >)

After a week we decided to not do 2 separate prototypes. Although we understand that the separate prototypes serve to try different things, we have 2 reasons. We would really love to do a roguelike and also quite big time constraints and we would be behind if we started the roguelike later. So I got to thinking. The first and biggest thing in the roguelike is the procedural environment. I am doing this for the first time so doing a completely generated dungeon would be hard. We also want some control of the rooms we make. Obviously there is still control in complete generations. But for use it would be easier if we create the rooms and then use them to create a grid like dungeon, where there would be one room in a grid cell with corridors leading to other rooms.

So once again I am pained by the time constraint. We have to do a prototype pretty much every week and that doesn't give the time necessary for me to dive into the algorithms one would use for something like this.



I created a very basic dungeon generation. There are multiple rooms created with corridors already built. So there are rooms with combinations of corridors on the right, left, top and bottom. Each room has a spawn point on each side there is an opening. And there will spawn the next room from an array that has rooms with this kind of opening.



This would spawn a random room with an opening on the right

So while this works and creates a pretty nice dungeon of what I want there is a bug.

Sometimes there is a room spawned either going into nowhere or a wall going into a room that doesn't have an opening there. Also it generates rooms until it can't because it is bound to spawn closed rooms. Obviously because the spawner is not checking for this kind of thing. But this is all I could do this week.

Week 3 (15.10. >)

I sort of fixed the bug. I unfortunately had to write a new script for checking if there is an opening to the rooms. I did it by placing wall checks on the edge of the colliders and checking if there is a different wallcheck object. If there is it does nothing. If there is not it replaces the current room with a room that is closed and has only one opening. This kind of opening only works because all the rooms have maximum of 2 corridors. This wall check script is bigger than the room spawner.

I had a revelation while working on this bug. The way I have the dungeon spawning set up is very very inefficient. It doesn't allow for any easy expanding, let's say for some special rooms to spawn, or just controlling easily what rooms spawn. I realized I would have to rewrite the dungeon generation after the prototyping phase ends in a better way. Currently I have to keep it like this because I have to work on the other prototype iterations. I have to again emphasize on the lack of time we have for quite big things. I would love to experiment and figure out stuff myself but it's impossible.

I also used this bug fixing to make a limit to how many rooms spawn instead of until it can't spawn anymore. So the arrays containing the rooms don't include the closed rooms anymore and instead it checks how many rooms are spawned and if there is more than the limit it will not spawn. And since the wall check is there it will close up the the open rooms, so it's nice that it combined like this into a different problem.

Also this week is for the topic Random Obstacles not just bug fixing. So basically the way we want it is that when the player enters a room that is a combat encounter random enemies spawn in random positions.

And here comes the combat loop we thought about. It will essentially be a bullet hell where the player mainly focuses on dodging all the projectiles that there are flying against him. The attacking will be that it automatically shoots at the closest enemy. Why? The game will already quite hard by itself and if we add manual shooting on the top of dodging a lot of bullets it might be a lot. However for the attacking there is fortunately more time to experiment in the future.

Also we thought about the health system. At least to me it is quite important because it is a bullet hell with focus on dodging, therefore focus on not being hit. So having a basic health bar I feel like is just disappointing. I thought about having just health points, basically hearts, and when the player gets hit it will decrease by one point (heart) and put emphasis on the hit so some kind of flashy animation.

I was thinking of Hollow Knight while thinking of the health system and because we want the player to heal somehow we can add also the charge healing. Where there is a container and it fills up whenever the player hits something and when the player want's to heal, he will hold a button it will stop the player in place while it's charging and after a while it will heal the player by one point.

This is something that will definitely make the game pretty hard but to me it seems like something that benefits game where there is a focus on not being hit and I presented it so to my team.

Mike who unofficially joined our team, created a simple enemy on my request. It is a very simple stationary enemy who constantly shoots at the player in range. It serves for the current prototype and time constraints.

Week 4 (22.10. >)

This week we got Sophie and Mike (Antonio Lazarov, officially now) on our team. Both great additions, Sophie who wants to be more on the design side of the game and Mike on programming who will lessen my load a lot.

So while did make a functioning enemy script it was made for 3D, so it didn't shoot at the player who is 2D. He did it on purpose, apparently because he was once told that making a 2D game but still with 3D objects eliminates a lot of problems mainly with collisions but no concrete examples were given. I asked Hagen who substituted this week and he said he doesn't see a reason to do so and just use 2D for 2D games. Mike then agreed to change the script for 2D.

I talked with Hagen, how I could go about doing the upgrades for the random abilities. We talked for a while and basically pointed me in the direction of Scriptable Objects so I have to look at those and see if I can do them this way.

Also I created the basic combat loop of shooting at the closest enemy.

Now that we have a design person on our team (Sophie) we can have some good input. She said that the health system is depending on how hard we want to have our game. I also asked her to create some ideas for upgrades for this week. I also talked with her a little when

asking about what she has come up with. And it does seem like she might make us more grounded because she did push back on asking us on more concrete stuff we're able to do before she can come up with the upgrades. So it's great to have someone like that. And to think I doubted her skills before, I was proven wrong.

I created a simple Upgrade system. I went with Scriptable Objects, something Hagen told me about. Unfortunately since I didn't have that much time, I couldn't implement the choice to pick which upgrade to get from as a reward. So instead I made something I already know and it drops the upgrade on the death of an enemy.

I quickly found out how beneficial Scriptable Objects are, to create a lot of similar but different objects and store data. So I created a Scriptable Object template (abstract) and then made specific upgrades that inherits from the main template. Then I made an object that drops on the death of an enemy and is like an empty container. On death it just sets a random upgrade from an array of the upgrades and sets some stuff like the name.

Although while this works I have no idea if the way I set it up is a great way and I have to ask Jorg about it.

Week 5 (29.10. >)

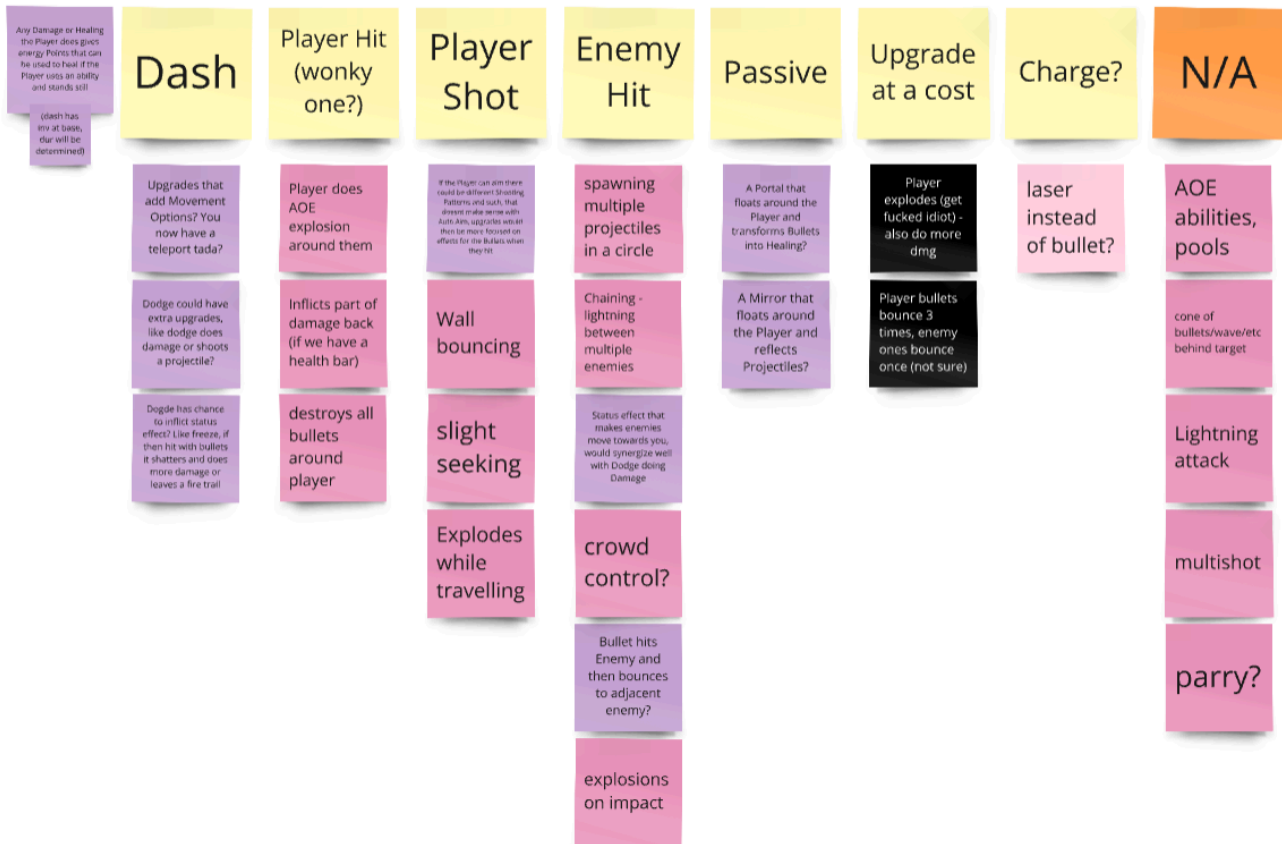
In this week I did 2 main things. The health system and the shooting modes. Because Mike insisted that he would like to have manual shooting instead of auto shooting, instead of deciding on one mode I implemented so that the player can toggle the between the auto and manual shooting modes and we decided that we will let people try this out on the playtesting evening and see what they like.

I also created the health system and luckily I was successful. It is how I described, player has health points, when the player gets hit with a bullet the time stops to simulate a hit stop. Because there is no animation right now the game just stops for a second. I also am curious how people like it.

One big thing that concerned me a little also was the scale of the sprites. Because Unity Tilemaps scales down the sprites to 100x100 it made the tiles created by Clara so much smaller. We were struggling to figure out how to go about it also with the idea of how to make it as modular with the procedural generation. As of right now the procgen is not that modular and I am planning it on rewriting it. But we are struggling to figure how much the camera sees, how big dante is and how big the rooms/environment tiles have to be.

On Monday, the day before playtesting evening, we had a meeting. We mainly discussed the gameplay and the upgrades.

Slots for Upgrades



(sorted by priority)

We thought of an upgrade system similar to the one of Hades. Where on the action of a player there would something happen. The player can choose what upgrade he has in each slot and replace them. Basically the whole meeting we brainstormed what upgrades the player could have and what sort of actions there could be performed by the player. Basically at the end we came to the conclusion of having these actions (viz. image above). When player dashes, when he gets hit, when he shoots, when an enemy is hit, a passive, risky upgrades (high risk, high reward), maybe a charge ability (low priority), and lastly upgrades that we might implement if there is time. the charge is also undecided because there is already a lot of actions the player can do and we don't want it to be confusing.

Vertical Slice

Week 1 (5.11. >)

This is the first week of the Vertical Slice that I think is also like a transition week because we officially haven't in class started the vertical slice. But in this week there is the Playtesting Evening.

This week we had a guest lecture from a guy called Luca about his roguelike game he is developing and he also gave us feedback on our game. I would boil down the feedback to saying that our game just 'exists', it is just another roguelike and there is nothing special to it. After he brought that forward, I realized that it is so. Our game is just a combination of different games. We are lacking a 'thing'.

Playtesting Evening

There are 3 main questions we had for the players trying our game.

Question 1: How do you like the shooting? Do you like auto aim or do you like manual shooting?

Question 2: What do you think about the health system, do you like the health points system and what do you think about the hit stop.

Question 3: What do you think about the camera? Is the player too small? Is the camera too far?

The people generally liked the auto aim more because they like to focus on the dodging more and it made the dodging more fun. My idea about having the auto aim was similar to that. However, at one point Eyal pointed out that if there is auto aim and there is no ammo mechanic when is there ever a reason to not hold down the button and not shoot. So even if the consensus is that auto aim might be better it gave us a lot to think about. That if we want shooting we might have to think about some resource management but that is another unnecessary difficulty on top.

The health system was well received, even though I didn't make the healing in time. One thing the people really didn't like was the hit stop. Although they understand it is to show that they got hit, they just didn't like they lost control for so long, partly because there was no animation so it felt like the game just froze.

And the camera, we already struggled with the size and scale. However Florian gave us some advice. He didn't like how our rooms are big and that the player is pretty small because of the camera. He gave us advice that we should start with smaller rooms and see how our camera fits with the amount of enemies and how much we see. But generally people thought that the player is pretty small and some had issues with the hitboxes,

however, those were conflicting. Some thought that the player is too big so dodging was hard but some people thought that the player is small so the dodging was easier.

One thing that gave me a piece of mind or even made me really happy was that Luca, the guest lecturer, told us that our team is probably the most solid of the teams that had the presentations. Even though it was a small comment, it made me happy and validated some of the work I have so far done.

Because our game just felt generic we already started thinking about what to do about it. I don't know who suggested but a potential idea was not having the shooting at all perhaps focusing on the movement/dashing more. That was certainly an interesting idea, because it would solve a lot of design problems we are having right now. Like, the game being generic shooter we would focus on the bullet hell part of the game and make it a little unique. It would solve the problem of choosing a shooting mode. And it could create some interesting combat. However, we were too tired to decide on anything and too soon to decide.

The Big Change Meeting

We had a meeting a few days after the Playtesting Evening about the feedback and how we should continue. The main problem was that our game just felt too generic. We also had a problem with the shooting. After a lengthy debate we have decided to throw out the shooting completely. We decided like this, so that we completely focus on the bullet hell aspect and make the game more about avoiding projectiles. Instead we would focus our combat system around the dashing ability. This would fix two of our problems, the problem with deciding on a shooting mode and the problem of having our game be generic. We then tried to brainstorm how to use the dash as a combat mechanic, not only just as a movement mechanic. Some proposed having the combat encounters just be surviving until a certain amount of time passes. However I countered, because having this just be the whole gameplay loop would be incredibly tedious and boring after a while. I think I proposed damaging enemies by dashing through them, however Sophie said this would just be replacing a ranged attack with something similar to a melee attack.

In the end Sophie brought forward a pretty nice idea. When the player dashes through an enemy it will mark them/apply a mark and after a while it will damage the enemy or do something. This kind of thing is pretty nice because we can make the abilities for this and make some pretty nice synergies with it. For example when the player dashes through an enemy and marks them it will apply a status effect that gives overtime damage, that also might apply a slowness effect. Or an ability that creates an explosion once the mark ends combined with an ability that makes the mark end once the player dashes through the enemy while the enemy is marked.

Another very big change we decided on is to completely throw out the procedural dungeon aspect of the game. While it is something that would be nice to have we have a couple of reasons. First of all, I would have to redo the script so that it would allow for easier modification not just from designers but also for further enhancing of the script. I wouldn't have to then spend so much time on this and could focus on something else that is also

important, like the combat system. Other reason, it would allow us to do more stuff thematically. Of course we don't have to completely throw it out, we could make multiple versions of the map and just randomly choose which one the player will spawn in. This is a big change that alters our game in a major way.

Week 2 (12.11. >)

This week I did a minimal amount of work because there were a lot of events happening like Talk & Play and Games Ground, so I attended and volunteered on both of those. After that I was completely tired to do something. Nonetheless, I still finished the healing system so the player can now heal himself, we still haven't decided what will fill up the gauge now that the shooting is gone, so for now it's unlimited healing.

Week 3 (19.11. >)

I added the healing mechanic. Admittedly it is very similar to Hollow Knight's healing system but I believe this is something that fits for this game. It is risk based healing, you have to hold a button and wait a while to heal by one heart. It makes the player think when to heal during combat and reinforces the bullet hell aspect.

I also transitioned to the New Input System and learned a lot how to set up the code for that.

The main thing I created this week was the Mark system. My aim while doing this was to make it as flexible as possible so I can easily add new abilities, and make it work with Scriptable Objects so that that the abilities can easily be changed. It will function similarly to the Hades boon system. This week I am working only on the mark itself and the marked dash system where it executes an ability when dashing through an already marked enemy. There are 2 very important scripts excluding the abstract scripts of the abilities. The `PlayerUpgradesManager` and the `EnemyMarkHandler`.

```

public class PlayerUpgradesManager : MonoBehaviour
{
    public MarkBehavior equippedMark;  ⚡ Default Mark Behavior.asset
    public OnDash equippedOnDash;  ⚡ Unchanged
    public MarkedDash equippedMarkedDash;  ⚡ Changed in 0+ assets
    public Passive equippedPassive;  ⚡ Unchanged
    public float markDuration;  ⚡ "8"

    2 usages
    public void ApplyMarkToEnemy(GameObject enemy)
    {
        EnemyMarkHandler markHandler = enemy.GetComponent<EnemyMarkHandler>();
        if (markHandler != null)
        {
            markHandler.markedDashAbility = equippedMarkedDash;

            markHandler.ApplyMark(equippedMark, gameObject, markDuration);
        }
        else
        {
            Debug.LogWarning("The enemy you are trying to dash doesn't have the EnemyMarkHandler script");
        }
    }
}

```

All the Manager does is store the abilities and apply the Marks to the enemies. I wanted one script to be the center for the stored abilities for ease of use.

```

🔥 Event function  ⚡ usages  ⚡ overrides  👤 Simon SkvaraPC +1  ⚡ extension methods
void Update()
{
    if (currentMark != null)
    {
        Elapsed += Time.deltaTime;

        currentMark.OnUpdate(gameObject, Time.deltaTime);

        RefillSoul();

        if (Elapsed > markDuration)
        {
            currentMark.OnExpire(enemy:gameObject, player);
            ClearMark();
            return;
        }
    }
}
}

```

1 usage Simon SkvaraPC

```
public void ApplyMark(MarkBehavior newMark, GameObject player, float duration)
{
    if (currentMark == null)
    {
        //apply mark if there is no mark
        ApplyNewMark(newMark, player, duration);
    }
    else
    {
        //execute the marked dash effect if there is a mark
        MarkedDash();
    }
}
```

1 usage Simon SkvaraPC +1

```
private void ApplyNewMark(MarkBehavior newMark, GameObject player, float duration)
{
    this.currentMark = newMark;
    this.player = player;
    this.markDuration = duration;
    this.Elapsed = 0;
    this.playerHealth = player.GetComponent<PlayerHealth>();

    currentMark.OnApply(enemy:gameObject, player);
}
```

1 usage Simon SkvaraPC

```
private void MarkedDash()
{
    Debug.Log(message:"Attempted to do the marked dash", gameObject);

    if (markedDashAbility != null)
    {
        markedDashAbility.Execute(enemy:gameObject, player);
    }
}
```

The EnemyMarkHandler is where most of the mechanics happen. It stores the current mark and calls on the method that all marks have. And here come the abstract Scriptable objects

```

public abstract class MarkBehavior : ScriptableObject, IAbility
{
    public string markName;  ⓘ Changed in 0+ assets
    [TextArea(1, 3)]
    public string markDescription;  ⓘ Changed in 0+ assets
    public Sprite markIcon;  ⓘ Serializable

    public GameObject markPrefab;  ⓘ Changed in 0+ assets

    ⓘ 0+3 usages ⓘ Simon SkvaraPC
    public string AbilityName => markName;

    ⓘ 0+2 usages ⓘ Simon SkvaraPC
    public string AbilityDescription => markDescription;

    ⓘ 0+2 usages ⓘ Simon SkvaraPC
    public Sprite AbilityIcon => markIcon;

    /// <summary>
    /// Called when the mark is applied
    /// </summary>
    /// <param name="enemy">The enemy the mark is on</param>
    /// <param name="player">The player that applied the mark</param>
    ⓘ 1 usage ⓘ 2 overrides ⓘ Simon SkvaraNB
    public abstract void OnApply(GameObject enemy, GameObject player);

    /// <summary>
    /// Execute the mark behavior as if it was Update
    /// </summary>
    /// <param name="enemy">The enemy the mark is on</param>
    /// <param name="deltaTime">Time.deltaTime</param>
    ⓘ Frequently called ⓘ 1 usage ⓘ 2 overrides ⓘ Simon SkvaraNB
    public abstract void OnUpdate(GameObject enemy, float deltaTime);

    /// <summary>
    /// Called when the mark expires
    /// </summary>
    /// <param name="enemy">The enemy the mark is on</param>
    /// <param name="player">The player that applied the mark</param>
    ⓘ Frequently called ⓘ 1 usage ⓘ 2 overrides ⓘ Simon SkvaraNB
    public abstract void OnExpire(GameObject enemy, GameObject player);

```

Here is the MarkBehavior, it stores abstract methods that each ability will have. Since I want

it as flexible as possible I added three methods. They each function as if it was a Start method, Update method and the OnDestroy basically. I pass necessary variables in there but also sometimes the player if it is ever needed.

Here is an example of the Default Mark and how it works.

```
public class DefaultMarkBehavior : MarkBehavior
{
    public float damagePerTick;  ⚙️ "20"
    public float tickInterval;  ⚙️ "2"

    private float timer;

    private GameObject activeMark;
    private Animator animator;

    0+1 usages  👤 Simon SkvaraPC +1
    public override void OnApply(GameObject enemy, GameObject player)
    {
        Debug.Log(message: $"{markName} was applied to {enemy.name}", enemy);

        activeMark = AttachMarkToEnemy(enemy);

        enemy.GetComponent<EnemyMarkHandler>().activeMark = activeMark;

        animator = activeMark.GetComponent<Animator>();
    }

    ⚡ Frequently called  0+1 usages  👤 Simon SkvaraNB
    public override void OnUpdate(GameObject enemy, float deltaTime)
    {
        timer += deltaTime;
        if (timer >= tickInterval)
        {
            enemy.GetComponent<EnemyHealth>().TakeDamage(damagePerTick);
            timer = 0;
        }
    }

    ⚡ Frequently called  0+1 usages  👤 Simon SkvaraNB +1
    public override void OnExpire(GameObject enemy, GameObject player)
    {
        Debug.Log(message: $"{markName} expired on {enemy.name}", enemy);

        //Destroy(activeMark);
    }
}
```

I also made the MarkedDash, that is significantly simpler.

```
public abstract class MarkedDash : ScriptableObject, IAbility
{
    public string abilityName;  🐞 Changed in 0+ assets
    [TextArea(1, 3)]
    public string abilityDescription;  🐞 Changed in 0+ assets
    public Sprite abilityIcon;  🐞 Serializable

    📄 0+3 usages  👤 Simon SkvaraPC
    public string AbilityName => abilityName;

    📄 0+2 usages  👤 Simon SkvaraPC
    public string AbilityDescription => abilityDescription;

    📄 0+2 usages  👤 Simon SkvaraPC
    public Sprite AbilityIcon => abilityIcon;

    📄 1 usage  📄 1 override  👤 Simon SkvaraNB
    public abstract void Execute(GameObject enemy, GameObject player);

    📄 0+1 usages  👤 Simon SkvaraPC
    public void AssignToPlayer(PlayerUpgradesManager manager)
    {
        manager.equippedMarkedDash = this;
    }
}
```

Over all I am extremely happy that I managed to do this and that it works. My head almost exploded while making this system but it paid off.

Week 4 (26.11. >)

I added two new ability types and that is a OnDash (when player dashes) and a Passive. Here are the scripts

```

public abstract class OnDash : ScriptableObject, IAbility
{
    public string abilityName;  ⓘ Changed in 0+ assets
    [TextArea(1, 3)]
    public string abilityDescription;  ⓘ Changed in 0+ assets
    public Sprite abilityIcon;  ⓘ Serializable

    ⓘ 0+3 usages
    public string AbilityName => abilityName;

    ⓘ 0+2 usages
    public string AbilityDescription => abilityDescription;

    ⓘ 0+2 usages
    public Sprite AbilityIcon => abilityIcon;

    ⓘ 1 usage ⓘ 1 override
    public abstract void Execute(GameObject player);

    ⓘ 0+1 usages
    public void AssignToPlayer(PlayerUpgradesManager manager)
    {
        |   manager.equippedOnDash = this;
    }
}

```



```

public abstract class Passive : ScriptableObject, IAbility
{
    public string passiveName;  ⚡ Changed in 0+ assets
    [TextArea(1, 3)]
    public string passiveDescription;  ⚡ Changed in 0+ assets
    public Sprite passiveIcon;  ⚡ Serializable

    0+3 usages
    public string AbilityName => passiveName;

    0+2 usages
    public string AbilityDescription => passiveDescription;

    0+2 usages
    public Sprite AbilityIcon => passiveIcon;

    /// <summary>
    /// Apply the effect of the passive
    /// </summary>
    1 usage 3 overrides
    public abstract void ApplyEffect(PlayerUpgradesManager manager);

    /// <summary>
    /// if it is a passive that modifies the player stats, use for any clean up
    /// </summary>
    1 usage 3 overrides
    public abstract void RemoveEffect(PlayerUpgradesManager manager);

    1 usage 3 overrides
    public abstract void OnHitPassive(GameObject enemy, GameObject player, PlayerUpgradesManager manager);
}

```

In the passive there are 3 methods because for normal stat based passives you need to somehow go back to the original state if the player gets a different passive. And the on hit passive was specially implemented for the MarkChain ability (gives the mark to any enemy within the radius of the enemy currently being marked), however it can be used for other stuff.

Now, we actually want the player to have the option to choose randomly an ability. The procedural generation of this project. Now because the scriptable objects are all of different types and I didn't want to have huge hard coded scripts to handle each case I had to find a different way. Here I found/rediscovered interfaces. I never used them so it was interesting to use, but essentially I understood they can be inherited however many times a script wants. So I created a interface that would function as a global identifier and had every ability inherit it. I then added special method AssignToPlayer that would individually handle to which ability/variable they would be assigned.

I had 2 scripts, one that is essentially just taking care of the UI, displaying the information and one that is taking care of which abilities to even display.

Here is the interface.

15 usages 11 inheritors 0+5 exposing APIs

public interface IAbility

{

3 usages 4 implementations

string AbilityName { get; }

2 usages 4 implementations

string AbilityDescription { get; }

2 usages 4 implementations

Sprite AbilityIcon { get; }

1 usage 4 implementations

void AssignToPlayer(PlayerUpgradesManager manager);

}

Example of the AssignToPlayer

0+1 usages

public void AssignToPlayer(PlayerUpgradesManager manager)

{

manager.equippedMark = this;

}

This is the script for the UI.

```

public void Initialize(PlayerUpgradesManager manager, GameObject rewardChest, PlayerMovement movement)
{
    upgradesManager = manager;
    chest = rewardChest;
    playerMovement = movement;
    playerMovement.StopMovement();
}

```

```

/// <summary>
/// Shows the option on the selection menu
/// </summary>
/// <param name="ability1">The first random ability</param>
/// <param name="ability2">The second random ability</param>

```

1 usage Simon SkvaraPC

```

public void ShowOptions(IAbility ability1, IAbility ability2)
{
    option1 = ability1;
    option2 = ability2;

    //set name of abilities
    option1Name.text = ability1.AbilityName;
    option2Name.text = ability2.AbilityName;

    //set description of abilities
    option1Description.text = ability1.AbilityDescription;
    option2Description.text = ability2.AbilityDescription;

    //set sprite of abilities
    option1Image.sprite = ability1.AbilityIcon;
    option2Image.sprite = ability2.AbilityIcon;

    option1Button.onClick.AddListener(call: () => SelectOption(option1));
    option2Button.onClick.AddListener(call: () => SelectOption(option2));
}

```

2 usages Simon SkvaraPC More...

```
private void SelectOption(IAbility selectedAbility)
{
    selectedAbility.AssignToPlayer(upgradesManager);
    Debug.Log(message: $"Chosen: {selectedAbility.AbilityName}");

    playerMovement.ResumeMovement();

    Destroy(chest);
    Destroy(gameObject);
}
```

1 usage Simon SkvaraPC

```
public void CancelSelection()
{
    Debug.Log(message: "Canceled Selection");

    playerMovement.ResumeMovement();

    Destroy(chest);
    Destroy(gameObject);
}
```

And this is the script for choosing the abilities. It is on the chest that spawns from completing a combat encounter.

1 usage Simon SkvaraPC

```
private void OpenChest(PlayerUpgradesManager manager, PlayerMovement movement)
{
    IAbility ability1 = GetRandomAbility(manager);
    IAbility ability2 = GetRandomAbility(manager);

    // ensuring they are not same
    while (ability1 == ability2)
    {
        ability2 = GetRandomAbility(manager);
    }

    GameObject uiInstance = Instantiate(abilitySelectionUI);
    AbilitySelection selectionScript = uiInstance.GetComponent<AbilitySelection>();

    selectionScript.Initialize(manager, gameObject, movement);
    selectionScript.ShowOptions(ability1, ability2);
}
```

3 usages Simon SkvaraPC

```
private IAbility GetRandomAbility(PlayerUpgradesManager manager)
{
    IAbility ability = null;
    int attempts = 0;

    do
    {
        ScriptableObject randomAbility = container.abilities[Random.Range(0, container.abilities.Length)];

        if (randomAbility is IAbility validAbility && !manager.IsAbilityEquipped(validAbility))
        {
            ability = validAbility;
        }

        attempts++;
    }
    while (ability == null && attempts < 10);

    return ability;
}
```

It is very nice to find out about interfaces because thanks to them the selection works quite smoothly.

The other thing that is important for the procedural generation is the random spawn position of enemies in the combat encounters. So initially it was a square spawn area and it was really easy getting a random location inside a rectangle area, but since we went away from the square rooms the spawning became more complicated. Initially, Mike (Antonio) said he would use the mechanics he uses for the enemies to move and that we would get a random position from the waypoints that are spawned. However, he later said that is not viable anymore. Honestly, thank god Sophie said we can define an area using Polygon Colliders because I went to research and found out I can store the points of a Polygon Collider. Now I

will disclose that this following piece of code was not done by me because the final math was too much for me, but at least I understand most of it. What I am doing here is retrieving 3 random points from the collider and creating a triangle and then retrieving a random point withing that triangle.

```
private Vector2 GetRandomPosition(PolygonCollider2D collider)
{
    Vector2[] localPoints = collider.points;
    Vector2[] worldPoints = new Vector2[localPoints.Length];
    for (int i = 0; i < localPoints.Length; i++)
    {
        worldPoints[i] = collider.transform.TransformPoint(localPoints[i]);
    }

    // a random triangle (assumes the polygon is convex)
    int randomIndex = Random.Range(0, worldPoints.Length - 2);
    Vector2 p1 = worldPoints[0];
    Vector2 p2 = worldPoints[randomIndex + 1];
    Vector2 p3 = worldPoints[randomIndex + 2];

    // random value within the triangle - Barycentric
    float u = Random.value;
    float v = Random.value;

    if (u + v > 1)
    {
        u = 1 - u;
        v = 1 - v;
    }

    Vector2 randomPoint = p1 + u * (p2 - p1) + v * (p3 - p1);
    return randomPoint;
}
```

The only downside of this is that it only works for convex polygons, but it is enough for our intentions. If we had more complicated rooms, I would have to somehow make it work with non-convex polygons.

Overall we busted our asses this week, because of the upcoming Vertical Slice deadline.

Vertical Slice deadline (3. 12.)

So today we had the Vertical Slice presentation. To say the least it was a disappointing result. Although we did a lot, the result was lacking. Although Jörg had a lot of valid criticisms it felt incredibly disappointing not having validation for the hard work we put in.

Admittedly, we are making an action game and we didn't have feedback for the action. Also

the combat feels very slow, I think that partly might be because of how the enemies move and maybe the slow bullets.

The thing is, we already knew about those, but we don't have any time to do that with all the different projects we also have to work on. I suppose we will have to downscale even more to finish this game.

Production

Week 1 (3.12. >)

This week was a 'cooldown' week. We worked a lot for the Vertical Slice deadline and wanted to rest a little bit and work on the other classes. Everyone in the team agreed to this so there wasn't much work done during this week.

Week 2 (10.12. >)

I have discovered a major flaw in my ability system. Whenever I put a mark on multiple enemies at the same time the marks interfere with each other, making it so that it will not give damage as it is supposed to each enemy that has a mark, but it only gives damage to one enemy at a time even if there is multiple enemies marked. And that is because the ability is made through a Scriptable Object. I don't know I could have been so stupid, when it is so obvious. There is only a single instance of it and it is not initialized like a normal script would be, and therefore when I am referencing something or decreasing/increasing a timer all off the mark interfere with this one instance.

So while I have been able to so far make it flexible and readable, to fix this quickly I unfortunately made it more confusing. To be clear, I only had to change how the mark works the other abilities I didn't have to touch. However, I had to change the EnemyMarkHandler. I had to make it a per-enemy basis. Basically the handler script has a dictionary in it and whenever I mark an enemy it will log the enemy and a struct MarkState that has all the information for the mark and add it into the dictionary.

In the ability scripts themselves, I then get a reference to the dictionary and manipulate the values in there.

It is incredibly confusing but it works. I am very upset I didn't realize this problem sooner, because I would have done it a different way. The one upside to this is that at least, it is easy for game designers to balance the abilities, since they are nicely done in a scriptable object and they just add those to ability pools however they want. I am incredibly upset though, how it turned out in the backend.

Also there was a nasty Git problem where a bunch of work somehow got deleted in a merge, thankfully Benny helped us to resolve the problem.

Week 3 (17.12. > + Holidays)

During this period I didn't do much except making some abilities and fixing bugs. We however had the playtesting evening and we had some feedback. Mostly it was lot of quality of life and small things that they would like to have in the game to feel better.

Obviously it is an action game and we need it to feel better. We were still lacking in visual communication however it was much better now that we also had a light up for when the enemy is damaged. I also used the same effect for when the player heals. They also had a lot of balancing 'complaints' that we did take to heart to make it feel better. It is genuinely insane how much the playtesting evening can help, since whenever I am testing out something myself I get used to it and don't see all these small mistakes.

There was a suggestion to put the mark that is attached to enemies somewhere else than the center of the enemy, so that it is visible who is marked. That is a good idea that even Clara brought up before, however I don't think we implemented that, I don't know why.

Perhaps it was lack of time? Or it was that we forgot since we had so much to do.

What is also such a conundrum is keybinds. Every time there is a playtesting evening, there is always someone who doesn't like the keybinds. Even Mike would like it differently. The problem with that is that everyone has different preferences. We could empirically analyse the best control scheme and have that as default but everyone has small preferences. All that could be solved by adding rebinding. However, we have little time and a lot of stuff to do so I couldn't do it. We will just recommend to play with controller.

I planned on working on some small stuff during the holidays, however that fell through since I tasted the feel of relaxing and I didn't want to stress myself with the project.

Week 4 (7.1. >)

This is the week right after holidays where we realized that we only have a week left until the deadline and not 2 weeks. We realized we might be screwed. Luckily we managed to extend the deadline by 3 days.

During this week I finished the abilities I have yet to do. However mostly I wasn't creating 'content' for the game anymore. My occupation was to add the 'game feel', like sounds and screen shakes.

So for that I made a simple audio system using the Audio Mixer in Unity. It is not perfect since the volume is controlled through decibels and they function exponentially, meaning that sliding the volume bar halfway basically mutes the sounds. However that is enough for the time we had. I figured out how to make spatial audio for a 2d game and it is surprisingly easy. Normally audio listeners are on the camera and that would be bad since spatial audio is for 3D and the camera is away from the plane. So I just put the audio listener on the player. Since I wasn't doing anything content wise I was sourcing sound effects. Who knew it could be so hard to find sound effects for the stuff we need. I downloaded so many free sound packs off of Itch. Luckily I found acceptable ones and now it feels very nice having some sound feedback to the player actions. The hardest one to figure out was the mark sounds. You want a sound effect that is prominent enough that you can hear it during combat, but you don't want it to be so annoying that it distracts the player. I think I however found some that are fine enough. For anything more we would need a sound designer.

Also at the end of this week we had the playtesting between classes with the Build a Toy class. Thanks to the new input system we had the game interactable with a controller. We found out that the game feels so much better with a controller. Why? Well the bigger control over direction I think is the main one. On a keyboard you only have 8 directions you can go in, but a controller adds much more to not only direction but movement as well. Next step is to figure out how to make the UI interactable with a controller too.

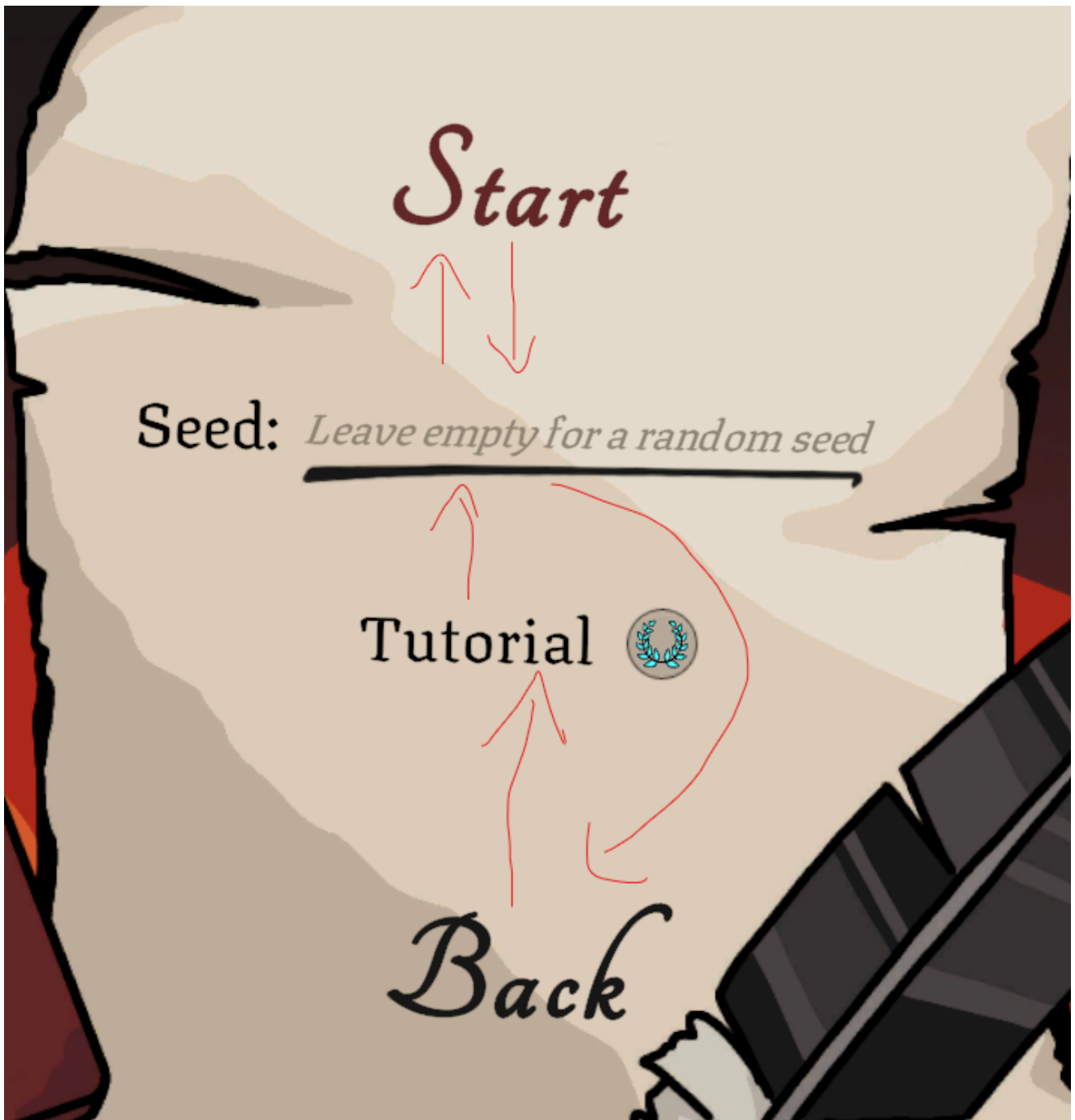
Final Stretch

The final stretch is the 3/4 final days that we got by extending the deadline. In here I mainly focused on the stuff that we needed to have like the Seed functionality or controller UI support and just generally tying everything together.

I initially thought that the seed functionality would be much harder to do, but there is a Unity Random method called `InitState` that does that for me. I just need to call it before any Random stuff happens and they will take that seed. I basically just then store the seed in a scriptable object, from the input field in the main menu and then in the main scene if the seed is 0 i generate a random seed and if it is not 0 i use that value. Also found out it doesn't have to be just positive but a negative `Int32` can be a seed too.

Next thing is making the UI interactable with controller. So while unity basically has the function for that already, there are a couple of caveats. I can add the first selected to the event system, but what happens when I change the menus, or open a different menu on top and then close it, now the selected button won't even be there. So I had to add to scripts to change the selected object anytime something like that happened.

However there is something that just grinds my gears. The Input field. While it is a Selectable when you go into it with a controller you are unable to leave and are forced to use Escape on keyboard. No button on the controller worked to leave the input field. So I had to make my own script, where it is possible to leave the the input field when I do something to change the selectable. That worked but here comes the infuriating part.



This would happen. Instead of going to the Toggle it would go to the button down there no matter what I would do. I even explicitly told the Selectable to go next to the Toggle I even explicitly referenced the Toggle in the script, yet for some reason it always went to the Back button. This bewildered me so much. I could come up with no reason what could make this. I eventually gave up because I spent too much time on it already and there was stuff to do.

Once I was done with this I helped with the Cerberus boss fight. Due to constraints we couldn't make it pretty but it was nice. There were supposed to be animations I was to implement but there was literally no time so there are only some simple looping animations. I also had to lock in at the end and bring everything together because the deadline was approaching and we had to have a win condition. Thanks to Clara we have a very nice UI and Thanks for Playing scene. We managed to upload the build 5 minutes before deadline.

And then we celebrated 🎉

We finished the game and we were quite proud of it as well.

Much to our disappointment, somewhere along the final touches some UI unreferenced and because of that the abilities weren't displayed and the ability selection wasn't closing. It is very sad that a bug like this got through like this. We all hope that Jorg got to play the version with this fixed.

But despite this hiccup we were proud to finish it.

Conclusion

To summarize the project. We definitely overscoped even though we want to do something within our range. That I think we failed. However in the end I think we created something that's at least somewhat fun. And one might argue that a fun game is a good game. But what definitely helps is the fact you can play it with a controller. It just adds so much to the game because it feels way better than the keyboard.

From a personal perspective, I learned a lot of coding. This semester I tried following proper coding 'rule sets' like separating scripts into components. Now that I am going further into coding and taking advice from Hagen's consultations I am enjoying coding more and more.

Overall, we made a game that I am proud of and even enjoy.