SMOL

Cyberspace Liberation - Computer-Network-Takeover-Operator delivers Net Neutrality
Triankolos Edition

Full Report

12.07.2021

Mehmet Dereli

Felix Kosian

Julius Krüger

Louis Hötzl

Game Proposal

Game Description

In our game the player is sent to “cyberspace” where they take direct control of a fixer and try to restore order. The game is presented in 2.5D and the player observes from top-down view while they move their avatar through level after level, shooting, dodging, and using abilities in order to defeat enemies until they arrive at the final boss. In classic rogue-like fashion the player has to restart from the beginning once their health drops to zero.

Aesthetically, in terms of sounds and visuals, we are aiming for an 80s Retro look (vaporwave) similar to Far Cry 3: Blood Dragon or the movie Kung Fury. Further, we want to use visual effects to reinforce the theme of chaos and order. While the player battles it out with the enemies the level is displayed in a chaotic fashion with distortions and a dark color palette. Once the player has cleared a level we want to change to a calmer color palette and remove the distortions.

1. Player

The player uses one set of controls to steer their characters movement and another to determine the direction in which they want to shoot as is common in twin-stick shooters. Since our player character is an agent of order we decided to have their abilities reflect that. As such the abilities are focussed on giving the player control over the enemies and the level. Potential abilities are:

- Dash: Allows to quickly reposition; the player character moves over a short, fixed distance in the direction of their current movement at a very high pace; the direction of the dash cannot be altered once the player committed to it
- Teleport: After a short channel time, the player character is moved instantaneously to a different point in the level within a certain range around the character; the player can pick the exact location they want to teleport to
- Slow Field: The player character lobs a grenade in a certain direction; once the grenade hits the ground it explodes into a circular field in which all enemies are slowed in movement and attack speed; the field disappears after a couple of seconds
- Hack: The player character hacks a standard enemy; the enemy character will then fight on the player’s side for a couple of seconds before switching sides again
- Charge: After a small wind up, during which the player cannot move or attack, the player will deal more damage for a few seconds

- Stun: The X enemies closest to the player are stunned for a few seconds
- Second Life: Once the player health reaches 0 they regain X% health instead of the game being over

Not all of these will be implemented. We plan to figure out which of them work and make for a coherent gameplay experience during prototyping. We might also consider giving the player control over which abilities they want to use, allowing them to choose between abilities, e.g., they can either use the dash or the teleport.

Further, the player can choose between different types of weapons. All of these weapons have low ammunition before they need to be reloaded. As such the player needs to be precise and make every shot count rather than implementing spray'n'pray tactics , thus further reinforcing a sense of order. The player can choose which weapon to use before each level, but they cannot change it afterwards.

- Pistol: Fires single shots, which deal a medium amount of damage each, with a medium delay between each shot, and medium amount of shots before reloading
- Burst Rifle: Fires bursts of three consecutive shots, dealing medium-low damage each, low delay between bursts, and high amount of shots before reloading
- Sniper Rifle: Fires single high damage shots, but needs to reload after every shot

2. Enemies

Enemies represent the element of chaos in the game. As such they move somewhat erratic, only semi-predictable, have fast-firing, inaccurate weapons, and have abilities that should disrupt or confuse the player. We aim to have different enemy types. Each enemy type has a fixed amount of health. An individual enemy is defeated once their health is depleted. Their goal is to bring the player's health to zero. Potential enemy types are:

- An enemy with a submachine gun type weapon; high fire rate, low damage projectiles; medium health
- An enemy with a shotgun type weapon shooting in a cone; individual shots deal low damage, but dangerous when up close; high health
- an enemy with a rocket launcher type weapon; has high range, but low fire rate; projectile explodes on impact dealing damage in a radius; low health

Additionally, we are planning on having a boss enemy as a final challenge for the player as well as a satisfying conclusion to the game. The boss should have very high health, but should also pose a challenge for which the player has to use the entire arsenal at their disposal rather than act as a simple bullet sponge, which is boring to play against, but takes ages to defeat. To accomplish this this final enemy should not

only employ different modes of fire between which it switches, but use abilities as well. Abilities could be:

- Creating mirror images of itself that distract the player, but do not deal or take damage; once hit they simply disappear
- Shockwave attack that deals damage if it hits the player character and knocks them away; during the knockback the player cannot move; the shockwave is not continuous, instead the player can dodge it at certain points

In addition to that the boss enemy should also occasionally spawn additional standard enemies to help it in the fight.

3. Levels

The game will consist of a series of levels, each of which consists of a single room. These rooms are bounded by walls and vary in size and shape. Rooms further contain different amounts and combinations of enemy types. In order to increase replay value we want to procedurally generate these rooms, varying size, shape, enemy combinations, and props in the room.

We further want to add traps to the rooms that can affect, both, the player and the enemies. Potential traps:

- Inversion area: Inverts the movement controls while the player moves through it
- Infection: The player takes damage over time
- Vortex: Knocks a character in a random direction over a medium direction; can be cut short by a collision on the way; player cannot move, but aim for the duration

We might add to the list or cut traps depending on time and ideas.

Once the player has cleared a level of all enemies the traps deactivate and the exit unlocks, which leads the player first to a sort of hubworld where the player can swap weapons and potentially change their abilities.

As the player progresses through the game, encounters become more difficult. To achieve this we want to increase the number of enemies the player has to fight during an encounter, combine different types of enemies for an encounter and, decrease the amount of cover provided to the player by obstacles in the level or increase the number of traps. Finally, after a number of levels the player encounters the boss enemy. If the player defeats that enemy the game ends.

4. Collectibles

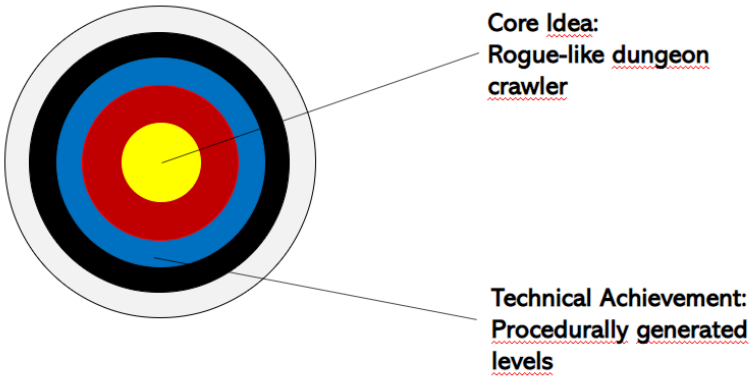
If time allows it we plan to implement collectibles which provide benefits to the player for the rest of their run. These will either be dropped by individual enemies

or spawn as reward for clearing a level. They will mostly be upgrades to certain weapons or upgrades to the player character's stats , i.e, more health, lower cooldowns, etc.

Technical Achievement

For our technical achievement we want to focus on procedural level generation. Levels will be generated such that they vary in shape, size and distribution of cover elements and props. Further, the amount and combination of enemy types and traps need to adapt as the player progresses in a way that smoothly increases difficulty, but also avoids any unfair situations in which the player is basically already dead the second they spawn.

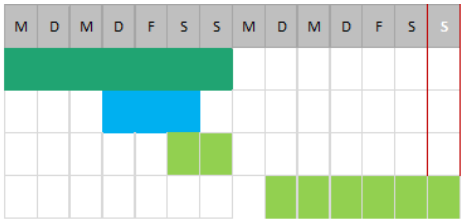
"Big Idea" Bullseye



Development Schedule

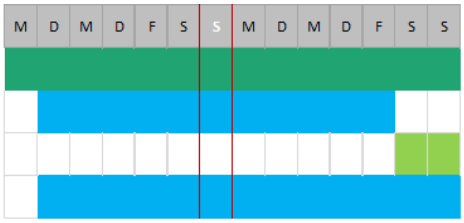
1. Game Idea

Milestone		Assigned to	Progress	Start	Days
Game Idea	Milestone			12.04.2021	7
Find Game Idea	Functional Minimum	All	100%	15.04.2021	3
Create Presentation	Deliverable	All	100%	17.04.2021	2
Write Report	Deliverable	All	90%	20.04.2021	6



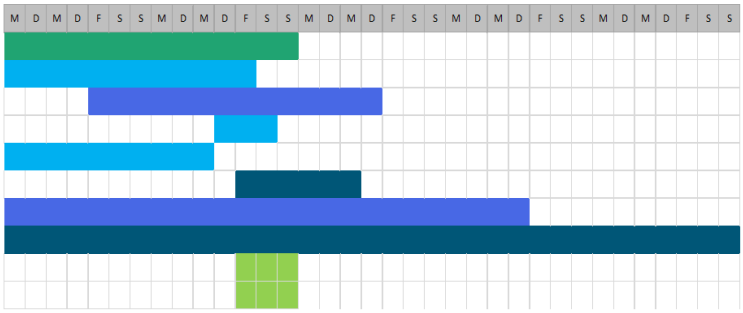
2. Prototype

Milestone		Assigned to	Progress	Start	Days
Prototype	Milestone			19.04.2021	14
Create Prototype	Functional Minimum	All	0%	20.04.2021	11
Write Report	Deliverable		0%	01.05.2021	2
Refine Game Design	Functional Minimum	All	0%	20.04.2021	13



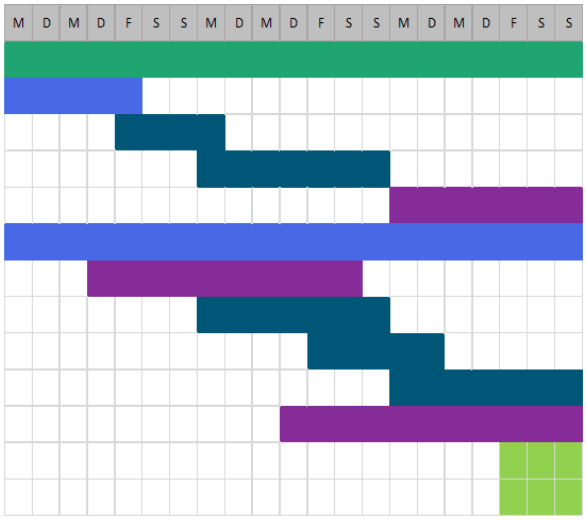
3. Interim Demo

Milestone		Assigned to	Progress	Start	Days
Interim Demo	Milestone			03.05.2021	14
Enemy AI: Basics	Functional Minimum	Felix	0%	03.05.2021	12
Obstacles	Low Target	Felix	0%	07.05.2021	14
Simple UI	Functional Minimum	Felix	0%	13.05.2021	3
Player Controller	Functional Minimum	Julius	0%	03.05.2021	10
Traps	Desirable Target	Julius	0%	14.05.2021	6
Procedural Level Generation	Low Target	Mehmet	0%	03.05.2021	25
Art and Sound Integration	Desirable Target	All	0%	03.05.2021	35
Write Report	Deliverable	All	0%	14.05.2021	3
Presentation	Deliverable	All	0%	14.05.2021	3



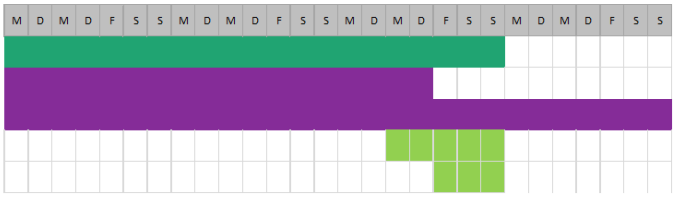
4. Alpha Release

Milestone		Assigned to	Progress	Start	Days
Alpha Release	Milestone			17.05.2021	21
Enemy AI: Boss	Low Target	Felix	0%	17.05.2021	5
Extended UI	Desirable Target	Felix	0%	21.05.2021	4
Additional Enemies	Desirable Target	Felix	0%	24.05.2021	7
Shader	High Target	Felix	0%	31.05.2021	7
Balancing	Low Target	Julius	0%	17.05.2021	21
Extend Player Controller	Desirable Target	Julius	0%	20.05.2021	10
Basic Narrative	High Target	Julius	0%	24.05.2021	7
Collectibles	High Target	Julius	0%	28.05.2021	5
Hub World	High Target	Julius	0%	31.05.2021	7
Visual Feedback Effects	High Target	Mehmet	0%	27.05.2021	11
Write Report	Deliverable	All	0%	04.06.2021	3
Presentation	Deliverable	All	0%	04.06.2021	3



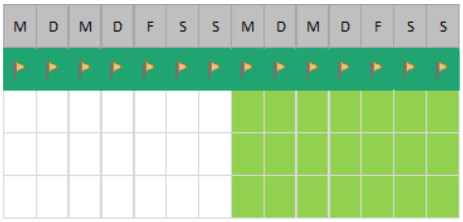
5. Playtesting

Milestone		Assigned to	Progress	Start	Days
Playtesting	Milestone			07.06.2021	21
Playtesting	Desirable Target	All	0%	07.06.2021	18
Implement Player Feedback	Desirable Target	All	0%	07.06.2021	28
Write Report	Deliverable	All	0%	23.06.2021	5
Presentation	Deliverable	All	0%	25.06.2021	3



6. Final Release

Milestone		Assigned to	Progress	Start	Days
Final Release	Milestone			28.06.2021	14
Write Report	Deliverable	All	**0%**	05.07.2021	7
Presentation	Deliverable	All	**0%**	05.07.2021	7
Video	Deliverable	All	**0%**	05.07.2021	7



7. Targets

a. Functional minimum
   i. Handmade level (Tutorial level)
   ii. One functional enemy
   iii. Basic Playable character
   iv. Simple UI (Health, Quit Game, Settings)

b. Low target
   i. Simple procedural level generation
   ii. Player abilities
   iii. Visually somewhat appealing
   iv. Boss enemy
   v. Level obstacles

c. Desirable target
   i. Fancier procedural level generation
   ii. Extended UI
   iii. More and animated enemies (~3)
   iv. Traps
   v. Sound effects and music
   vi. Well balanced

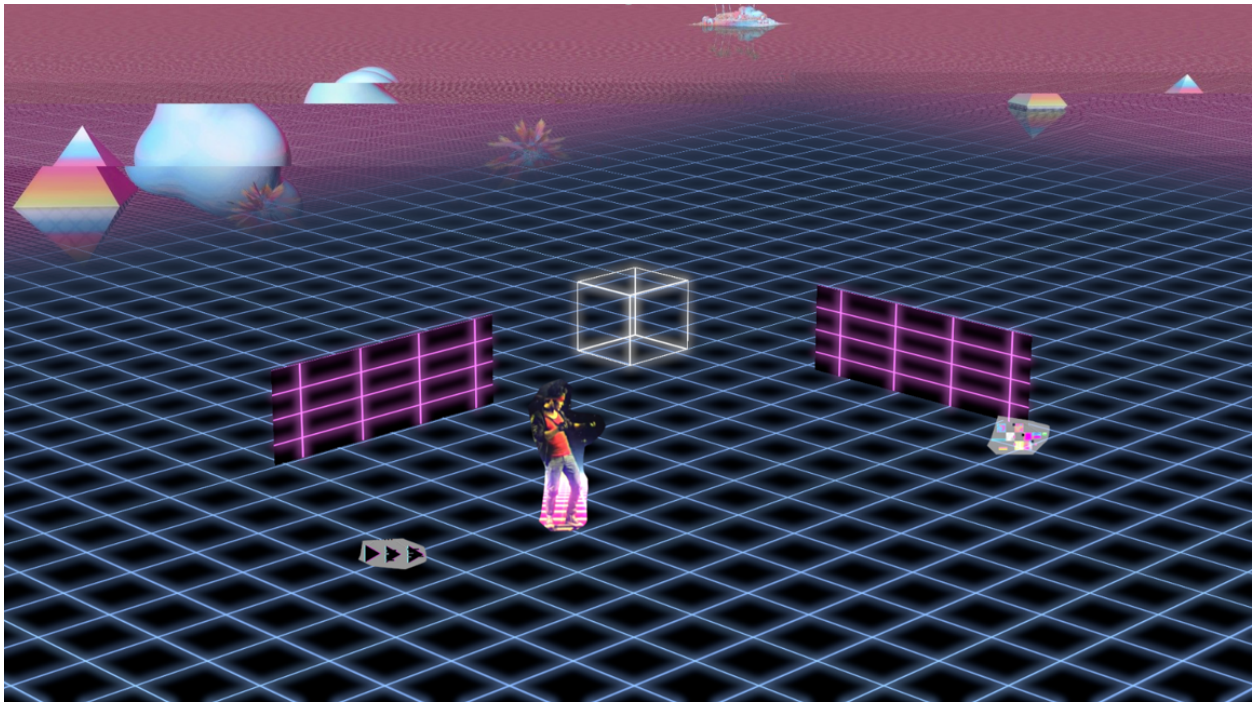
d. High target
   i. Visual Feedback Effects (i.e. screen shake etc)
   ii. More weapons
   iii. Collectibles (Player Power-ups/Upgrades)
   iv. Enemy drops
   v. Hub world
   vi. Basic narrative

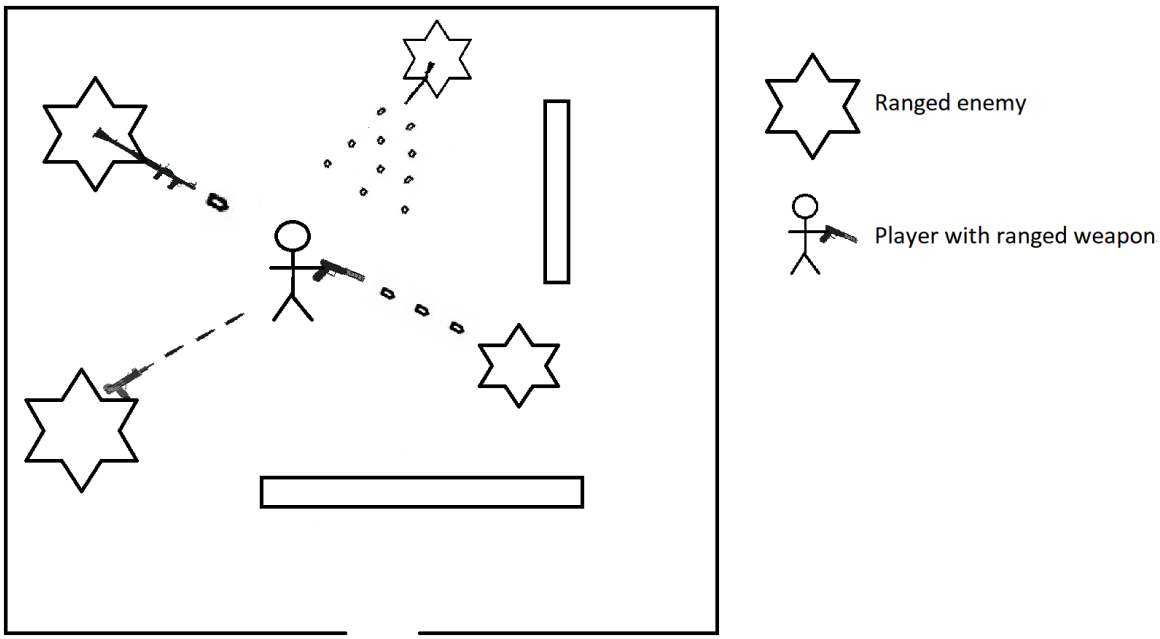
e. Extras
   i. Extended narrative
   ii. More enemies/bosses
   iii. Player progression system

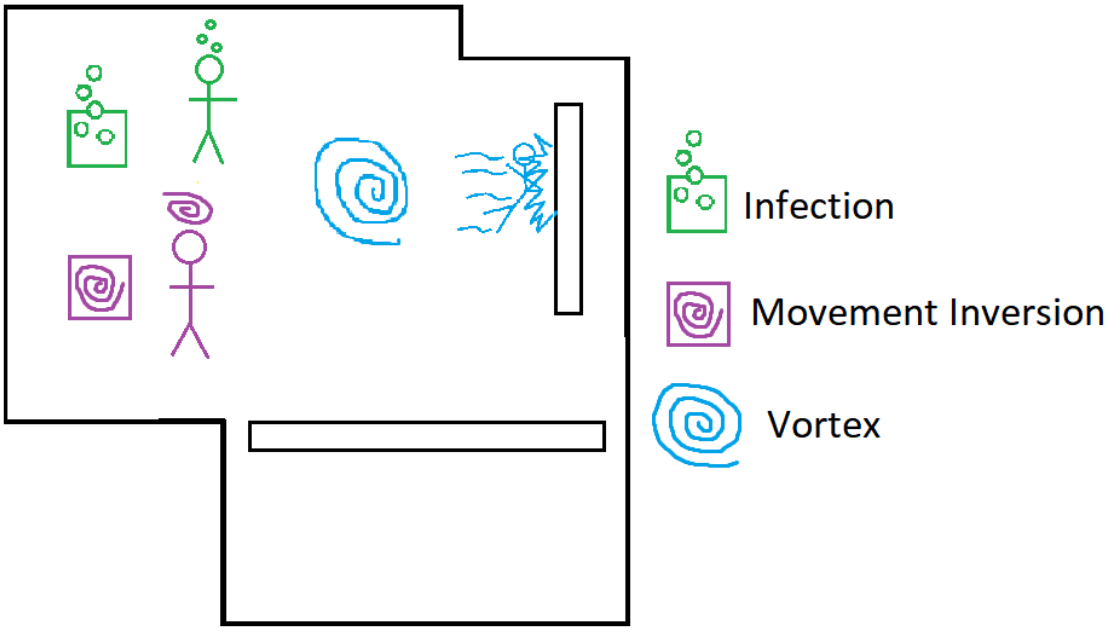
Assessment

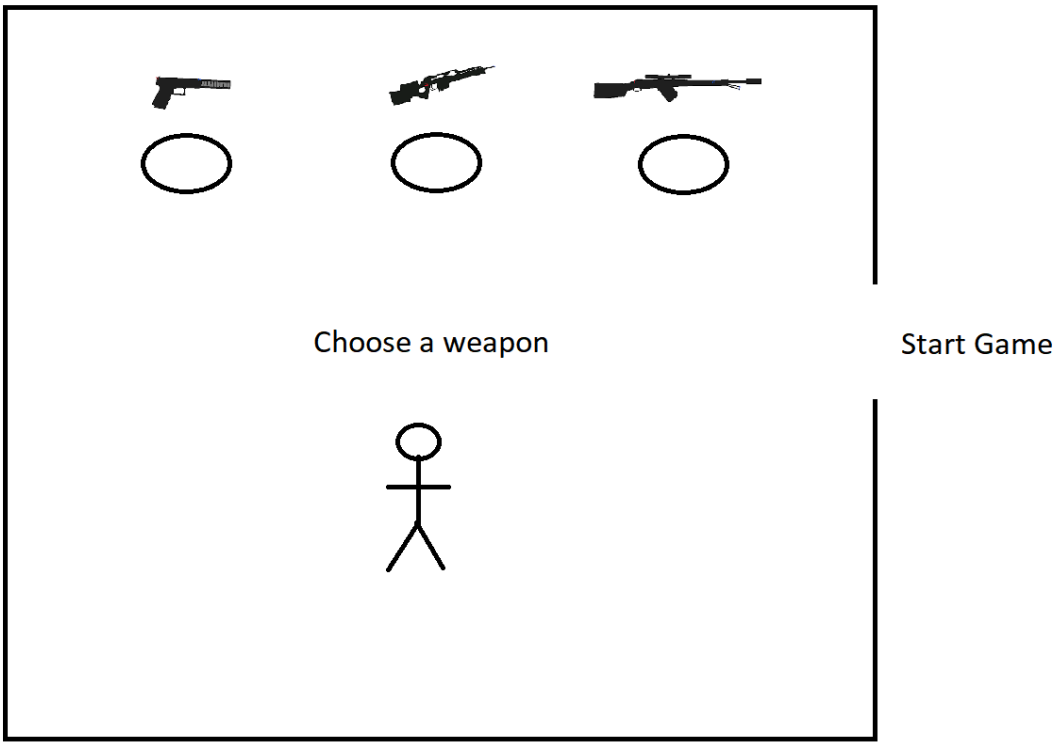
The game's main strengths should be it's well-balanced mechanics and it's juicy feedback when the player interacts with it. The game is mainly aimed at players who enjoy challenging themselves and are willing to restart the game over and over again to master the game's systems. Further, the game could appeal to players who enjoy the vaporwave aesthetics and are tolerant to failing a lot. We would consider the design a success if we manage to produce a well-balanced gameplay experience that never feels unfair to the player and keeps them engaged. Even when their character dies over and over again, the player always comes back to the game to try again.

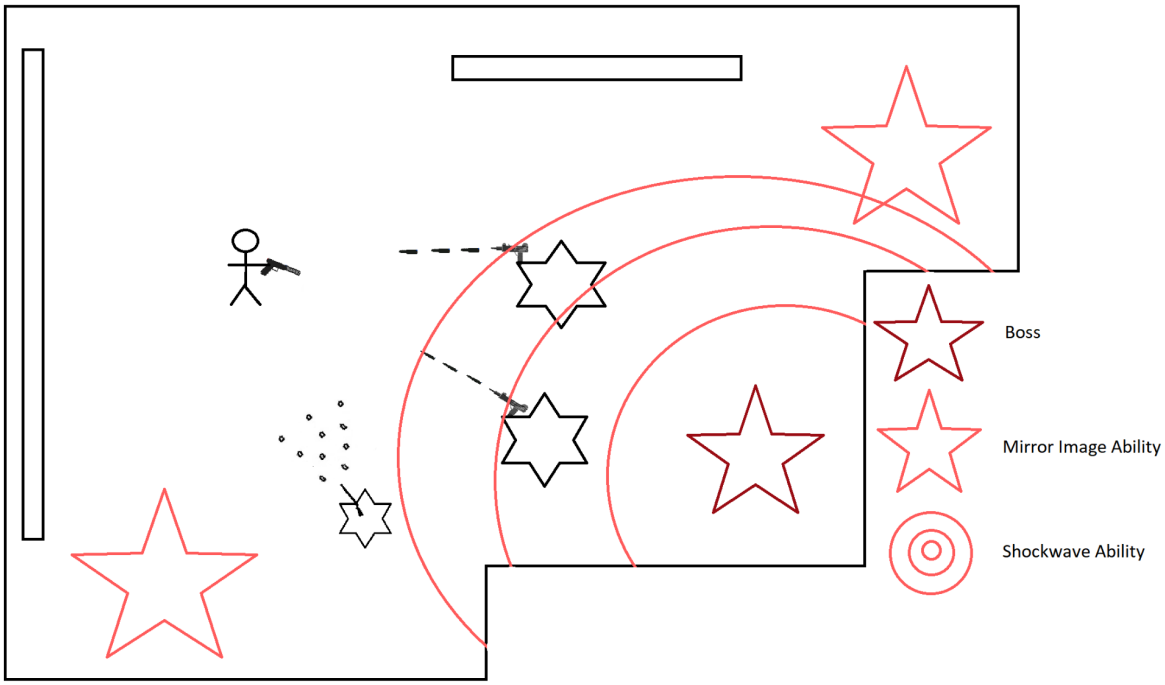
Sketches






Infection
Movement Inversion
Vortex


Choose a weapon
Start Game


Boss
Mirror Image Ability
Shockwave Ability

Prototype

Prototype Description

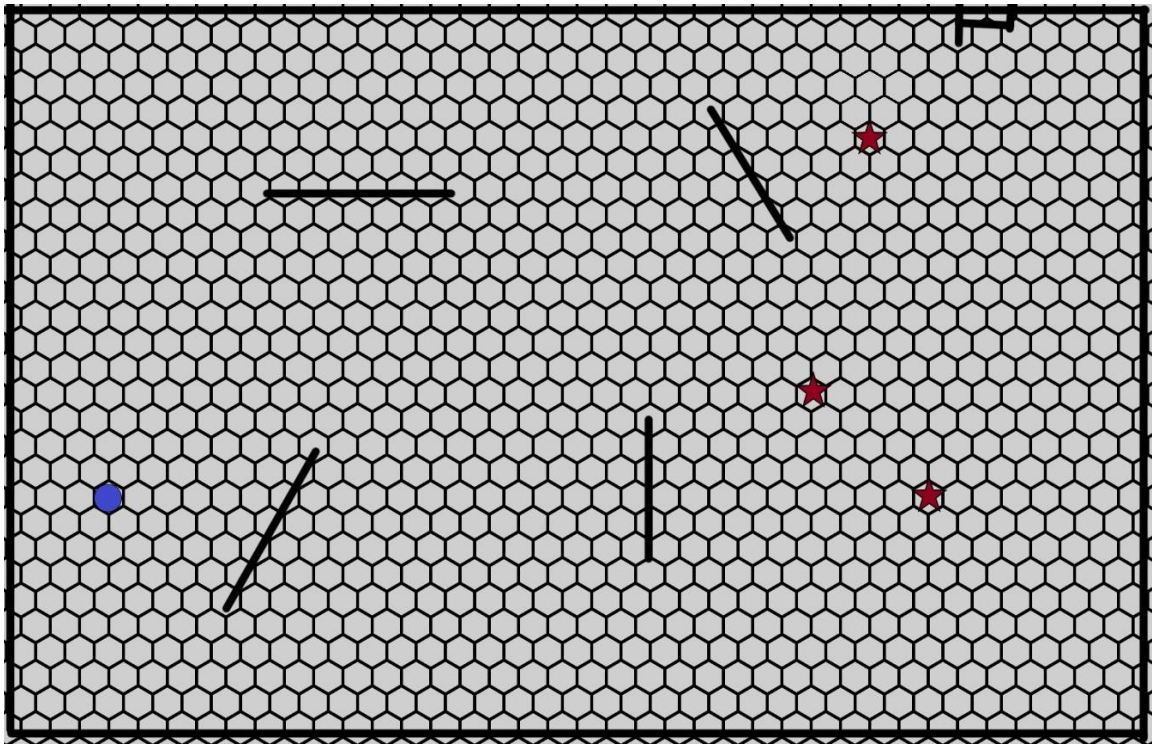
The first issue our prototype had to tackle was how to abstract the real-time behavior of the final game for an analog prototype. For that purpose we decided to use a turn-based approach in which each turn corresponds to three seconds of real time. We used two players to prototype the game, with one player being responsible for the player character's actions and the other taking control of the enemies. We decided on a fairly simple level design in terms of level shape and obstacles, since we didn't want to focus on this part of the game for the prototype. The enemy player gets a certain amount of points, starting at 30 points. With these points the enemy player can buy units. These units can then be placed in an area of the level opposite to the player spawn. After each level the enemy player gets 10 additional points to buy units with. Traps were placed somewhat randomly in the level, however, always in such a way that the player character would not instantly be killed once the game began. The player proceeds to the next level once all enemies in a level are defeated. The game is over once the player character defeats the boss enemy in the final level or their health reaches zero.
Since we opted for a turn-based approach for our prototype we also needed to somehow abstract our game space to make the prototype more manageable for the players. We decided to subdivide the space into a hexagonal grid. At the beginning of each turn we moved the bullets and after that the player character and the enemies took their turn simultaneously. During each turn every character can take a movement action as well as shoot bullets or use an ability.

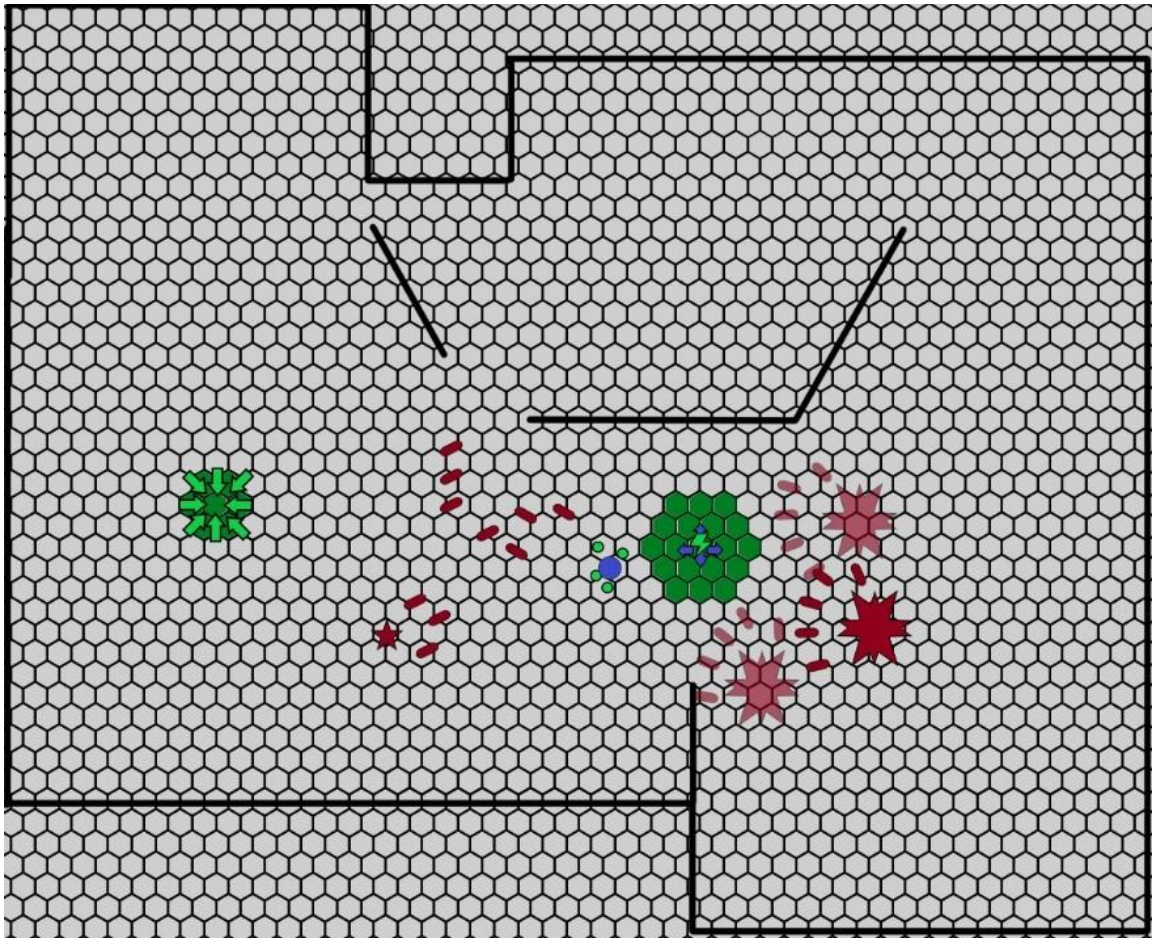
We decided on the following numbers to calculate game behavior:

- Character movement: 3 tiles/turn
- Player (Blue circle):
    - Health: 100
    - Weapons
        - Pistol: 10 dmg per shot, 6 bullets, max. 2 shots/turn, 1 turn reload
        - Burst Rifle: 3x5 (15) dmg, 21 bullets, 1 burst/turn, 1 turn reload
        - Sniper: 30 dmg, 1 bullet, 1 shot/turn, 1 turn reload
    - Abilities
        - Dash: 3 tiles range, 2 turns cooldown
        - Teleport: 15 tiles range, 1 turn channel time, 3 turns cooldown
        - Slow field: 3 tile radius, 2 turns duration, 3 turns cooldown, x0.5 attack speed and movement for enemies

        - Hack: 1 turn duration, 3 turns cooldown
        - Charge: 1 turn channel, 2 turns duration, 3 turns cooldown
        - Stun: 1 turn duration, stun 3 enemies closest to player
- Enemies
    - Submachine gun type enemy (red five-point star)
        - 20 health, 3 shots per turn, 20 dmg/bullet
        - 10 points cost
    - Shotgun type enemy (red six-point star)
        - 45 health, 1 shot of 3 bullets per turn, 15 dmg/bullet
        - 20 points cost
    - Rocket launcher type enemy (red eight-point star)
        - 65 health, on impact projectile deals 30 dmg in a two tile radius
        - 30 points cost
    - Boss enemy (red ten-point star)
        - 300 health
        - shoots 5 bullets in a cone, 20 dmg/bullet
        - appears after five levels
        - can spawn two reflections of itself, which mirror it's behavior
- Traps:
    - Inversion field:
        - Player movement is randomized while in the field (dice roll)
    - Infection:
        - 2 tile radius, player gets 3 dmg/turn while in the field, afterwards 3 dmg/turn for 3 turns
    - Vortex:
        - 2 tile radius, pulls player to the middle, then throws them 5 tiles in a random direction (dice roll)

For a further description of player and enemy abilities we would like to refer to our game design document ("Formal Game Proposal").





Prototype Experience

The main problem with our prototype was that due to the nature of our game a substantial amount of pieces, especially bullets, had to be moved every turn. This slowed down the pace of the game substantially. In hindsight, it might have been preferable to abstract the bullets as well. Further, it was very easy to mess up turns due to the chaotic nature of our game. More often than not, we forgot to move certain bullets or even characters or we just weren't sure whether they had moved already or not. Overall, we have doubts whether our prototype accurately reflects the gameplay experience we are going for.

Nonetheless, the prototype was at times quite tense, because the enemy player tried to predict player movement in order to defeat the player character. The prototype also helped us to better understand our design as it forced us to think about aspects of the game we probably would have thought about a lot later and also showed us some things that would have become very problematic later on.

What we have learned

The most important thing we realized during prototyping is probably that we need to reduce our scope if we want our final game to be a well-balanced experience. Although, some features may be relatively easy to add to the game, i.e. additional weapons are easily implemented after the first one, the effort required during playtesting and for balancing is going to grow exponentially.

Further, we realized that some of our planned mechanics were just boring and did not offer any interesting functionality. We were able to spice some of these mechanics up so that they contribute a little more to the gameplay experience. Others were just scrapped altogether to reduce the complexity of the game. We also have a first idea in what range player and enemy stats could lie, however, these will probably change a lot over the course of development.

We also realized that the enemy AI will be very important. The prototype was at times quite tense, because it was played by two humans. Our AI might need to be similarly challenging as a human who predicts and attempts to tactically engage the opposing player. Implementing this as an AI is certainly going to be a challenge.

Balancing of abilities will also be important. As it currently stands the abilities seemed very powerful. Depending on the AI this might make the game way too easy for the player.

Although we didn't focus on the level design aspect of our game in the prototype we learned that the level size will be important. In our prototype the levels were often too big

with a lot of empty space that was never used. This made the game feel calmer than we wanted.

Design Revisions

- For now, we are going to reduce the types of weapons the player can wield to one. Mechanically, there is not really a difference between the weapon types and as such the additional effort required to balance them can probably be spent elsewhere to a greater effect.
- A trap field (infection) where the player is damaged once they step on it is really boring. Instead, we decided to have the player chased by the infection for a fixed amount of time once they get too close. The damage is then applied if the infection catches the player character.
- In the prototype we did not test directed enemy movement with a random factor. We are not sure whether we want to keep that aspect of the game since the prototype worked without it.

Interim Demo

For the interim demo we have managed to implement a basic version of our game loop. The player can move through a procedurally generated level and shoot enemies, which shoot back. If the player dies or kills all enemies then a new level is generated. We have completed most of our goals for the functional minimum except for a few UI elements and although we do have a handmade level it would make for a poor tutorial. We hit most of our low targets as well. Our player character is still lacking a few abilities, however, and we do not have a boss enemy yet. We did implement a solid foundation for our AI that allows us to add new enemies with relative ease once we are set on their functionality.

Game Loop

Our game loop consists of the player and enemies spawning, fighting and when either all enemies or the player die a new level is generated. We handle this with a GameManager that organizes the game loop by resetting and starting a level. We use the manager to call the reset and start functions of the other managers to make sure everything is executed in the correct order. The current start order is level generation, navmesh building, AI and UI starting. Then player and enemies are placed in the level. Additionally it links independent subsystems together, for example the spawned enemies and their corresponding health bars.

Level Generation

The procedural generation of levels and their playability, is tightly linked to the player experience and therefore plays a vital role. In order to generate levels that are not only visually, but also structurally appealing, a lot of different aspects, such as complexity, density or size, come together and influence each other. None of those aspects are allowed to outweigh each other, because it would lead to a frustrating or boring player experience. If levels for example, are too big and complex, the player might find it frustrating to play those levels, but if they were too small and plain, the player might get bored. In order to balance those aspects, it is necessary to create a level generator with a lot of easily tweakable parameters, so that the generator can be adjusted rapidly during playtesting. Because this is not the first game with procedural level generation, there are a lot of different approaches to this task, such as Cellular Automata, Cave Generation or Chess Maze generation for example. These approaches however, did not appear to produce the results that we wanted, which is why we decided to take a different approach.

The levels in our game are based upon a 3D-CubeTileMap and consist of 3 different elements: The layout, the obstacles and the decoration. As of this moment, our interim demo includes 1.5 of these elements, which are the generation of the layout and a partial generation of the obstacles.

In order to reduce the computational cost of the algorithm, we are using a 2D-Array to store the information of each Grid-Cell and place the Tiles into the Grid afterwards. The level layout is generated by randomly generating different sized conjunct clusters of tiles and placing them randomly into our grid until a predefined threshold, which is either a minimum coverage of the level or a minimum number of clusters, is reached. The cluster sizes are randomly chosen from within a predefined range. Tiles that stick out of the level, can be either cutoff or kept, to allow for more choices during the playtesting. This method of generating levels however has a big flaw, and that is the possibility of disjunct spaces. This can be eliminated by choosing a small, high number range for the cluster sizes and a higher level coverage, but this will result in a lot of similar open field levels which might be boring. We decided to check if there are any such disjunct spaces by using a Flood Fill Algorithm and generate a new level if there are any such spaces. This might seem a little wasteful in regards to computational resources, but if the level generation parameters are set correctly, the chances of having disjunct spaces can be minimized.

After generating the layout, the walls are placed into the array. Because our algorithm only takes care of disjunct space and not of holes in the layout, the wall generation might place walls into the layout and thereby create obstacles. These walls will be included within the total number of obstacles that will be generated, so that the level generator does not have the possibility of creating too many obstacles.

Additional things that are generated include the PlayerSpawnPosition within the grid, as well as the EnemySpawnPoints. While the PlayerSpawnPosition is in the top-left corner of the map to ensure a certain level of consistency for the player, the EnemySpawnPoints can be anywhere on the map as long as they have predefined distances to the PlayerSpawnPosition. In order to avoid a single cluster of EnemySpawnPoints, poisson disc sampling is used. This method of generating points will also be used to create decorations and further obstacles.

One problem that occurred during the implementation process was the use of 3D Objects in combination with Unity's TileMap System. Because every object placed into the TileMap has to be a ruletile and those ruletiles do not support 3D objects, but 2D Sprites, a lot of features were not accessible for us in this project. One Problem is the use of objects that are supposed to be bigger than 1 Cube, because every object is automatically scaled to fit into that cube. If we, for example, want to use a statue as an obstacle within the level and its size is bigger than 1x1x1, we have to split the object into cube sized parts and rearrange them in the TileMap to fit them into the level. There might be other more suited solutions that we have not found yet, but the standard method of putting them into the grid does

not seem to work, which is why a roundabout way of solving this problem will be needed either way.

Character

We have implemented the character's basic functionality that is required for our game to be playable. As such the character can move, aim, and shoot based on the player's input. The character also has a health bar that is reduced when the character is hit by enemy projectiles, and dies when it is depleted. We have also implemented a dash that allows the player to quickly reposition in order to dodge projectiles or flank enemies. Initially, the dash was planned as a single dash after which the ability would be unavailable for a time. This didn't feel great and now we allow the player to make a second dash directly after the first one if they feel like it. Otherwise the ability is on cooldown.

Initially, we had planned to implement control schemes for mouse and keyboard as well as for a gamepad. This turned out to be more difficult than expected. The initial assumption was that Unity's Standalone Input System would offer us an easy integration of both input methods. The first problem we encountered was that for some reason Unity's system does not recognize the Right Stick of a gamepad. There is a workaround, but we decided to de-prioritize gamepad support for now as there is another problem with it. It would be relatively straightforward to add gamepad support for the basic elements (moving, shooting, aiming) of our controller. Some of our abilities are a bit more challenging, though. In particular, the teleport ability requires the player to select a location on the map to which they want to teleport. With the mouse this is a simple point-and-click action, but with the gamepad this action might feel a lot clunkier and will probably need some work to feel acceptable. We consider other parts of the game to be more important at the moment and as such will only integrate gamepad support if we have time for it.

AI

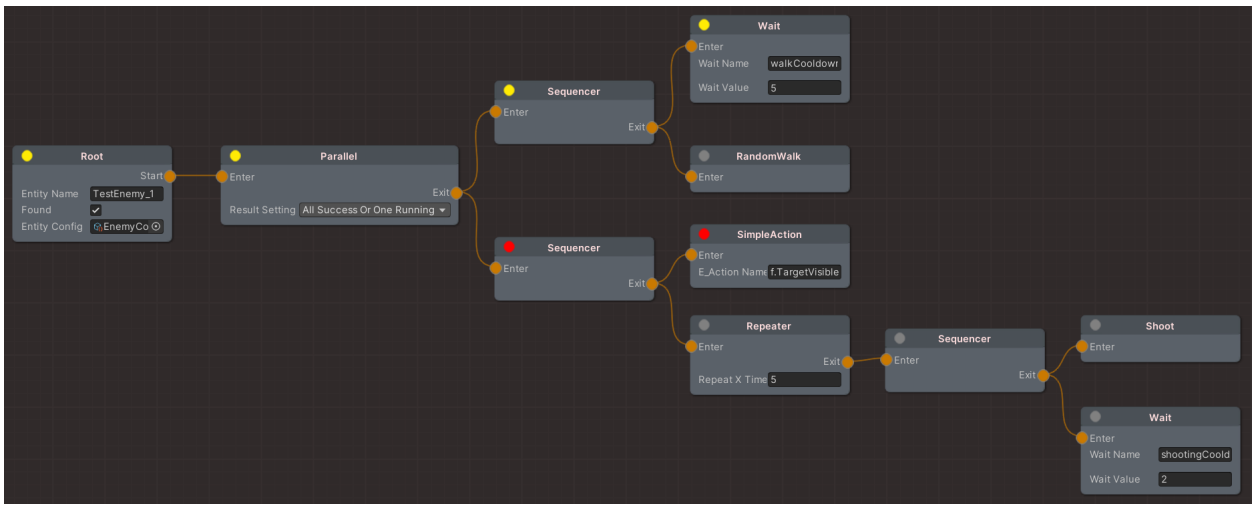
For an intelligent enemy behaviour we need 3 parts: 1. What can the enemy do, 2. How can the enemy interact with the environment, 3. How does the enemy know what to do and when. Furthermore there is an Ai Director which manages the macro actions like spawning enemies.

1. One type of enemy has a class which contains all actions the enemy can do. Currently those actions are: Check target visibility, shoot bullet towards target, take damage, die. One extra action not in this class is: select a random position to walk towards.
2. The main interaction with the environment is movement. This is handled by generating a navmesh after the level is generated. The enemy movement is handled by a Navmesh Agent Component which can navigate the enemy as close as possible to a target destination.
3. The intelligent decision making for entities is handled by a Behaviour Tree (BT). This is a

hierarchical structure which will run through it's nodes in a logical way defined by the game developers. It will be called every frame. Every node can return one of three results: success, running, failure. Based on those results, the tree can decide which node to invoke next. Nodes for the structure of the tree are Composite Nodes or Decorator Nodes. Composite Nodes can have multiple children and decide what to do with multiple results, for example invoke the children as a sequence or in parallel. Decorator Nodes can have one child and modify it's result, for example negate it or return "running" until the child returns multiple successes. Leaf nodes are actions which correlate to the enemy action with addition of general functions like "wait for x seconds".
Each BT can have multiple entities so the invocation of the tree is context related to reduce the amount of BTs. Information necessary for the entities is saved in a "Blackboard" which is a dictionary containing parameters and actions. Parameters are pre-defined in a serializable class and loaded into the blackboard at the start of the game.

At the start of the game a blackboard gets created and pre-defined parameters from a serializable class linked to the enemy and functions defined in the enemy behaviour are loaded. Afterwards the BT is created. Every frame the BT is invoked for each enemy.

To design and manage larger BTs a visual interface is needed. This is done with XNode (XN) which provides a simple node based visual framework. This XN Tree is read at the start of the game to create the BT. Each BTNode (which handles the logic) needs a corresponding XNNode (which handles the configuration).
For live debugging each BTNode saves its result for each entity in a separate dictionary so the XNodes can read them. Additionally an extra state "inactive" was added to the BTNodes to be able to see which BTNodes were active last frame.



Example of the XN Tree in the demo

In the example image the entity "TestEnemy_1" with a specific "EnemyConfig" is loaded. In this frame the BT is still waiting before it will move again and in parallel could not view the target and therefore didn't shoot.

Another feature of this implementation is live editing of entity configuration values (like shooting cooldown) directly in the XN Graph which is not fully functional yet.

Implementation difficulties arose while changing one BT to handle multiple entities as well as having multiple entity configurations in the same BT. An additional challenge was managing editor and runtime instances. While the XN Tree is present in the Editor as well as in Runtime (with a complete reload caused by the XN framework), the BT tree is only created at Runtime.

User Interface

Currently the UI consists of the player's health bar, three ability slots and the enemy health bars and is managed by the Ui Manager. The player's health bar is linked to the player's game object in the Game Manager over the Ui Manager. The enemy health bars are created in the UI Manager after the invocation from the GameManger.

To display the correct health information, we use UniRx with Reactive Properties which follow the observer pattern. As a result the entity health is directly linked to the Ui and updated live.

Design Revisions

We made the following design revisions:

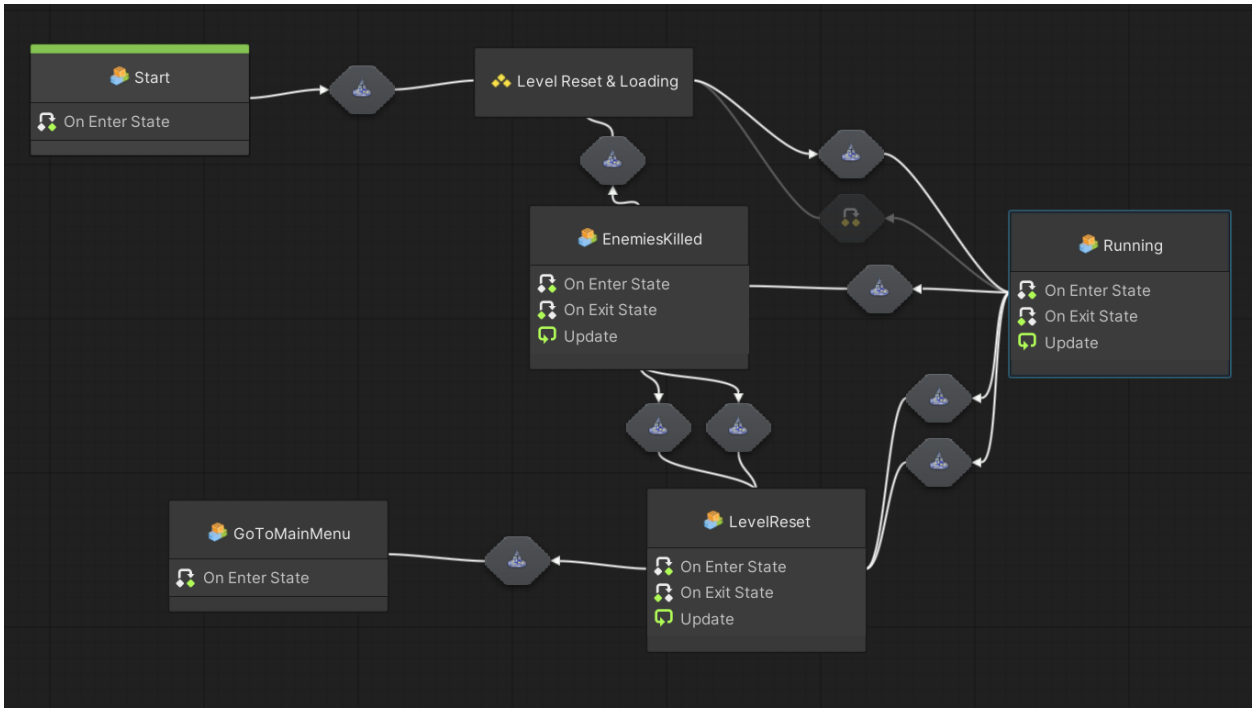
- Based on the feedback we received after the last milestone we decided to add a progression system to our game design.
- We also decided to reduce the area of the level accessible to the player over time by having level tiles break away into nothingness while the level is played. This should put an emphasis on speed. We also think this fits well into our setting. The more time the enemies spend in the level the more they corrupt it. It also acts as an interpretation of chaos, if chaos is to mean 'void'.
- The dash ability now allows for double dashes.
- We deprioritized gamepad support

Alpha Release

Since the interim demo, we came up with a proper title for our game and managed to implement complete a lot of the missing functionality that we had planned for our game. We finally completed the missing features of our functional minimum. Namely, we finished the UI and added a tutorial level. We also added a boss enemy and implemented the rest of the abilities that we had planned. This completes the low target as well. As stated in our desirable target we extended the UI, the procedural level generation always results in playable levels, and we added a trap. We also added an additional enemy, which gets us to a total of two enemies (boss not included) as opposed to our target of three total enemies. We did not have time, however, to integrate sound effects and music, as well as visuals for some of the abilities. In summary, we hit most of our goals for the desirable target. From our high target, we only got started on the progression system, but ultimately we did not have enough time to integrate the feature. In its current state, the game is also rather unbalanced, but we will tackle that during playtesting.

Game Loop

The Game Loop was changed to allow opening the door after a level is cleared, detect the player walking through the door and loading the correct scenes. To allow this more fine tuned Game Loop, we implemented a state machine which decides the correct behaviour.



Level Generation

The procedural Level Generation is one of the more important aspects of the game as it directly influences the player experience. Ever since the Interim Demo, we have made a few changes to the Level Design, including a rescaling of the tile and total level size.

One of the major changes was the thinning of the walls. During the ID (Interims Demo), we felt that having thick walls would make the player feel as if he were in a more enclosed space and strengthen the idea of an enclosed room. During the early stages of development, we discovered, contrary to our first beliefs, that it feels very uncomfortable to have the player move in such a closed room and that the player can vanish behind the thick walls. We initially did not think that it would feel that comfortable, because the player is always around the center of the screen, but it turned out to be a bigger problem. We therefore thinned the walls to counter those two inconveniences.

Another big change was the decision to break out of our grid structure. During our tests, we discovered that the levels did not feel as natural and comfortable to play as we expected. One of the major reasons was the inorganic spread of obstacles. We initially thought that by placing obstacles in an orderly fashion would further strengthen the aspect of order in contrast to the chaotic enemies. After a revision, we decided to move the obstacles into the theme of chaos and spread them more randomly. By doing so, we managed to incorporate both aspects of the theme (Chaos and Order) into the Level Design.

The next big change that we did was to incorporate 2 different level styles, Vapor Wave and Outrun. In the beginning we thought that it would be nice to be able to switch between those 2 styles when the player finished killing all the enemies, to symbolize the swap from Chaos (Outrun) to Order (Vapor Wave). The Outrun style uses a dark color palette, which is associated with Chaos and Evil, while Vapor Wave uses a bright color palette, which is associated with Order and Justice. Our group was rather split about this feature, because while it is a nice feature, a few of us felt that it would be a waste of assets. While fighting the enemy is something that takes time, passing through the door to get to the next level happens rather fast and we wanted to show off both styles as much as possible, to add more variables to the Procedural Level Generation. The main strength of this type of Level Generation is to generate as many diverse levels as possible to keep the player engaged and by adding more color pallets, the effect of diversity is further enhanced. In the end we went with the later option, but we decided to leave the first option open, in case it felt awkward in the final game.

The last change that we did was more of an addition than a change. We decided to add a Tutorial, that consists of a room with different pillars, which allow the player to change his abilities and which provide a short explanation of the abilities. This was done, so that the

player can first familiarize himself with the mechanics. We further added a dummy enemy that can be used to practice.

Character

Since the last milestone, we managed to implement the rest of the abilities we had planned for our game. To keep track of channeling times, durations and cooldowns we used coroutines for all of the following abilities.

We implemented a Teleport that allows the player to teleport to any spot on the map. Once the player enables the teleport the time at which the game runs is reduced by (currently) 30% and the player no longer controls the player character but instead controls the camera. The player can then pick any point on the map they which to be moved to and can confirm or abort the teleport with left or right click, respectively. The location the player is currently selecting is communicated to the player with a semi-transparent object and we indicate whether it is a valid teleport by changing the object's color to blue or red. We determine whether a teleport is valid or not by checking whether the location is part of the NavMesh we are using for our AI. Since we used the Unity Standalone Input System we thought we could simply reuse certain controls by having separate action maps for the Teleport and the rest of the character controller. This would allow us to, for example, to use WASD to move the character during most of the gameplay, but also to move the camera and not the camera while the player is in Teleport mode. For some reason, this did not work by enabling and disabling the corresponding action maps, however, so we had to add booleans to check whether the movement or teleport input should currently be used.

The Slow Projectile is similar to the regular projectile. However, it is shot at an angle and over a fixed distance. Once the projectile collides with any obstacle or the floor it stops moving and slows down all enemies that are in a fixed area around the projectile by changing the movement speed of the enemies within its trigger collider. Once an enemy leaves the area its movement speed is restored. The projectile is destroyed a few seconds after it hit the floor and a cooldown is started.

We also implemented a Hack ability that allows the player to take control of one enemy for a short amount of time. We get all enemies within a certain range of the player and then we change the target of the hacked enemy to the enemy closest to it. After a certain time, the enemy is released and targets the player again and the ability starts a cooldown. One restriction of this implementation is that the hacked enemy can only have one target at a time. However, we assume that players won't notice this during gameplay and therefore did not justify the additional workload that would have come with changing our AI system to allow enemies to have multiple targets.

We implemented a Weapon Boost. After a short duration during which the player cannot do anything, the fire rate and weapon damage of the player weapon is increased, while its

reload time is decreased for some time. Then the ability starts a cooldown. This ability is mostly tweaking values and starting a couple of coroutines.

The Revive on Death ability is pretty straightforward. When the player character's health is reduced to zero we simply check whether this ability is enabled and whether it has charges left. If yes, the player character's health is set to a certain amount and we subtract one charge. Otherwise, the player character dies and the game is over. The ability allows for a more defensive playstyle for players who want to take it safe.

We also implemented a feature that allows the player to switch between some abilities in-between levels. The player can now switch between Teleport and Dash, Hack and the Slow Projectile, and between a Weapon Boost and Reviving on Death.

We also implemented the character model as well as animations. We expected that the animations would take a lot of work due to past experiences. For the most part, the art integration was pretty straightforward. One minor hiccup that we had to deal with was that the shooting animation did not play as fast as the player could shoot without looking ridiculous. Therefore, we split the shoot animation into three parts and looped the middle part for as long as the time since the last shot was not above a certain threshold. With this fix, the animations work reasonably well.

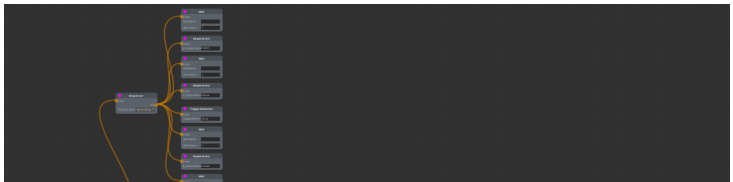
Additionally, we changed our camera behavior to be more dynamic. The camera is now moved toward the area into which the player is aiming rather than the camera being always centered on the player character. This turned out to be trickier than expected. The initial idea was to have the camera follow an invisible game object that follows the player's mouse pointer but only up to a fixed distance from the player. However, our camera is slightly tilted, and therefore objects at the bottom of the screen are closer to the camera's near plane than objects at the top. This resulted in the position of the game object being off and some weird camera behavior. Ultimately, we did not figure out the necessary vector math for the calculations, but we shot a ray from the mouse position in screen space into the world and then set the game object's position accordingly.

We also wrote a bunch of code for a progression system, but ultimately we did not have time to properly integrate it into the game or set up the necessary UI elements that are required so the player can actually interact with it.

AI

The AiDirector now manages difficulty and the special boss level. Each enemy can have multiple stages with difficulty cost tokens. Each level has a predefined amount of difficulty tokens. With two settings between 0 and 1 giving the mean, we can define the probability (following a normal distribution) of how many vs better stage and easy vs hard enemies will be spawned. Each spawned enemy reduces the amount of difficulty tokens left for this level.

For the boss level, the spawn



behaviour gets overridden and dynamic spawning enemies are supported.

Improvements to Behaviour Tree includes many more nodes (conditional checks, animation trigger, gameobject controls, ...) to allow much more complex behaviours.

User Interface

After a major rework, the UI now covers In-Game, Ability-Select and Menu. The Menu UI allows switching to different scenes (Game, Main Menu, Tutorial, Credits) and is partially available in all Scenes. The In-Game UI includes health bar, enabled and available abilities and ammunition count.

Design Revisions

We made the following design revisions:

- Thinning the Walls (see Section Level Generation)
- Rescaling the Tiles
- Breaking out of the grid structure by placing the obstacles more chaotically (see Section Level Generation)

Playtesting

Playtesters

We conducted a total of 15 playtest sessions with a mixed bunch of testers. Some of them had very little experience playing games in general, while others were quite experienced and some were even game developers themselves. Our testers came exclusively from our personal surroundings.

Procedure

During our playtests we first had the player play the tutorial level, followed by the main game while we watched them. Once the players decided they were finished with the game, we conducted an interview with the player with some guiding questions. We opted for an interview rather than a survey since we knew we would only get a rather limited number of testers and therefore it seemed to provide more valuable feedback.

We conducted some of our playtests in person, while others were done remotely. In the remote cases, we had our playtesters stream the game and we talked and listened to them via Discord. However, in the remote playtests we were not able to get other cues, such as body language, from our testers as not everyone had a webcam or sufficient bandwidth.

Observations and Feedback

Currently one of the largest issues with our game is the tutorial. The tutorial is very hands-off, does not properly explain some important parts of the game, and ultimately confused all of our testers to some degree. One problem was that when players entered the tutorial they were met by a prompt telling them how to move. We also had two enemies in the scene so the player could learn that they can shoot at enemies and that enemies die. Unfortunately, those enemies started shooting at the player the second they spawned into the level, which caused players to sort of panic. They expected a relaxed tutorial but felt they were immediately thrown into combat.

The next issue was that when we tried to teach the players about abilities, it is never explicitly explained that they are presented with a selection. Players then got confused that several abilities seemed to have the same button prompt. For some of the players, it became clear that they were presented with an ability selection once they entered the actual game and had the selection screen in front of them.

Our tutorial also resembled more of a sandbox in which players could try out the mechanics rather than a guided tutorial. Because of this, players were unsure whether the tutorial was over or whether there was something left to do. It also didn't help that there was no direct route from the tutorial into the main game. Instead, players had to return to the main menu and then had to start the game.

The next big issue that emerged during playtesting was our user interface. The color scheme of important UI elements, such as the player's health bar, is currently very similar to that of the UI background. Players, therefore, had issues finding the health bar in the first place and, once they had found it, it was problematic to keep track of it. UI readability was a problem in general. We had icons that indicate which abilities the player currently has selected and when those abilities are on cooldown we desaturate those icons. However, these icons are very small, so players had issues seeing them in the first place and the cooldown effect was even less visible.

The ability selection menu was also not as clear as it should have been. While some players understood that it was a selection screen, others thought that it was merely an overview of the abilities. Most likely this results from the visual difference between active and inactive abilities being miniscule. It was also not clear to players how to proceed from the selection screen to the level, because the 'Confirm' action was indicated by an 'X' button. On top of that many players had already forgotten what some of the different abilities did after the tutorial. It was also noticeable that those players that understood the selection, for the most part, didn't change their abilities after their initial selection. Some did another playthrough to try out a different set of abilities, however.

The feedback we got for our abilities was rather mixed. The dash, for example, was very well received, used a lot and players had a lot of fun with it. Therefore, the teleport was barely used and most of the players that used it thought it was inferior to the dash.

The hack ability was considered very useful and fun by some players, while others thought it was utterly useless. We think in part this was because the game does not communicate the range the abilities had. Therefore, some players thought the range of the ability was very low rendering the ability useless and others had a more accurate model of the range.

The slow grenade was rarely used as the game did not provide proper feedback where the grenade would land and the projectile also had a fixed range which made it even less useful. The weapon boost was considered by some players as way too powerful, while others thought it was too weak to justify being input locked for a second. The second life ability that revives the player on death was generally considered to be a good choice, but players never realized when the ability was triggered. In part, this was because the UI did not clearly communicate the player health bar and on top of that there is no visual feedback or animation that the ability was triggered.

Players also had mixed feelings on difficulty. Somewhat experienced players had no issues getting to the boss fight without dying and some of them even defeated the boss on their

first try. Those players thought the game was too easy and felt the game was dramatically less satisfying. Players with less experience playing games, or at least this type of game, struggled more with the game and often needed several attempts to defeat the boss. For some of these players, the game was too hard, but those that did eventually defeat the boss had a more satisfying experience for it. Many of our players also did not realize that the final fight was in fact a boss fight and thought it was just another type of enemy. In part, this was because the boss did not have a special health bar at the top of the screen or a name, which some players expected from a boss fight. The players also weren't expecting a boss fight already, since there were only very few levels leading up to it, and those were generally considered quite easy. Generally, players wished for more levels and for those levels to contain more enemies.

Overall, players enjoyed the game quite a lot and would have liked it to last longer. Many of them kept playing until they beat the game and many of those that easily beat it played it a second or a third time. Almost all of our testers thought that the controls were very smooth and most of them enjoyed the visuals and the music.

Some of our players also would have enjoyed more narrative framing at the beginning of the game and after they defeated the final enemy.

Feedback Gems

Some of the feedback we received was particularly interesting or inspired some ideas on how we could make our game more interesting.

First, the model of our boss enemy contains a sphere that is encased in a transparent pyramid. Two players thought the pyramid was a shield and they first had to defeat the minions spawned by the boss. They expected that the pyramid would then vanish and the boss would become vulnerable. Although that is not how our boss fight currently works, we think that it would make for a more interesting fight.

Second, we have one type of enemy that connects to other enemies of the same type with a laser. If the player entered that laser they would receive damage. However, the situation that the player entered the laser never arose. One player was unsure what the laser connection did. They thought it made the enemies stronger or maybe they had to destroy it in a certain order. We liked the idea that it would make the enemy stronger because currently, this certain enemy is fairly useless.

One player also suggested that we drop health packs at the end of a level so that players can regain some of the health they lose during the level.

Planned Changes

For the final release, we are planning to overhaul the tutorial so we can better introduce players into our game. To that end, we will break the tutorial into several sections, each of which will teach the player one component of the game and after that, the player is led directly into the main game. This way we can control the flow of information to the player better and can reduce stressful or overwhelming situations, such as enemies shooting at the player the second they enter the level.

We also need to rework various parts of the user interface. Most importantly, we need to make the health bar more readable to the player. The ability icons need to be larger and the cooldowns need to be communicated more clearly, i.e., with a top to bottom gradient that indicates how much longer the player needs to wait for the ability.

We are going to add a few levels leading up to the final level and tweak the difficulty so we have a nicer progression in our game. We are also considering adding a difficulty selection so inexperienced players and experienced players can have a similar gameplay experience if we have the time.

The abilities need to be tweaked as well. Some, such as the Hack and the Second Life, simply need to provide more feedback to the players. The slow grenade on the other hand needs to be reworked to a larger extend so it can be a useful ability.

We also need to add a winning and a losing screen to the game. Currently, the player is simply returned to the main menu. One player even thought the game crashed because of this.

If time allows we will also add some narrative flavor at the beginning and the end of the game to frame it a little bit better and provide a little more context to the player.

Conclusion

In Cyberspace Liberation Triankolos has used his army of minions to take control of Cyberspace. The player takes on the role of the Computer-Network-Takeover-Operator and is tasked with the liberation of Cyberspace.

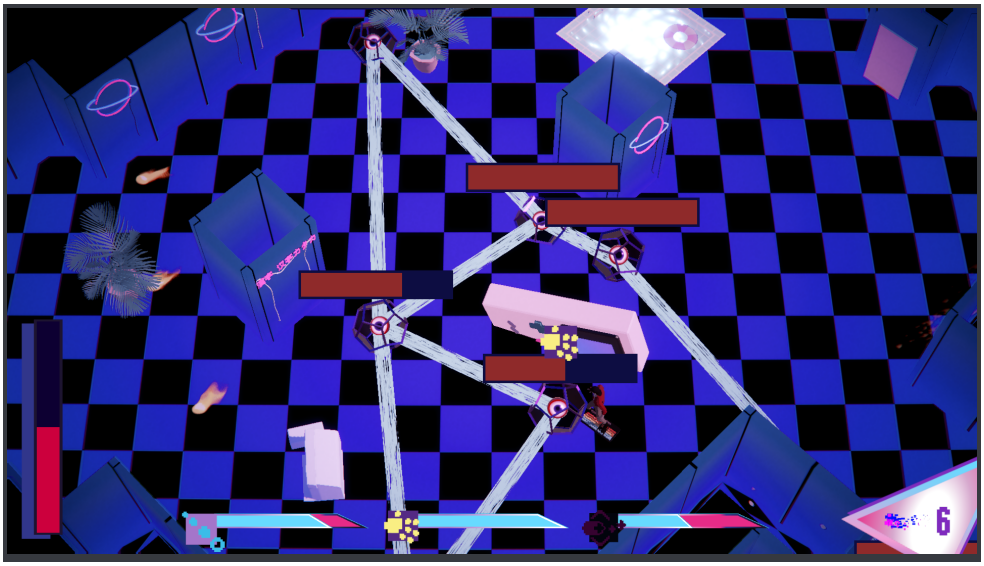


Fig. 1: In-Game Screenshot

To accomplish that the player has to fight through a series of procedurally generated levels to reach the boss. In classic rogue-like fashion, the player has to start over should they die at any point during their run to the boss. To give them an edge, they have a variety of abilities at their disposal, such as a dash, a hack, that allows them to take over an enemy for a couple of seconds, or even an ability that allows them to come back to life once, should they die.

Fig. 2: In-Game Screenshot

Fig. 3: In-Game Screenshot

Final Changes

For our final release, we have completely overhauled our tutorial. We now introduce the player gradually to the mechanics instead of throwing them into a level and overwhelming them with trying to understand abilities while being shot at. The player learns moving, then the abilities, and, finally, they fight two enemies before they are guided back into the main menu.

We have also redesigned the final boss fight based on player feedback. Simply spamming the shoot button until the boss dies does not offer much of a challenge. Instead, the boss

battle now has several stages in some of which the player needs to attack certain minions so the boss becomes vulnerable to the player's projectiles.

We also lengthened the game by adding a couple of levels in order to create a better emotional experience for the player. Instead of having relatively low tension and then spiking at the end with the boss fight, we now gradually ramp up the difficulty curve. On that note, we now also provide the player with a choice between an 'easy' and a 'hard' mode to make the game available to more players.

Finally, we have overhauled the UI to increase its readability and provide the player more easily with the feedback they need from the game, such as their health, or which of their abilities are currently available.

We also fixed a lot of bugs, reworked minor components of the game, and added some more content, but these were less significant than those mentioned in the previous paragraphs.

Fig. 4: Main Menu UI Before

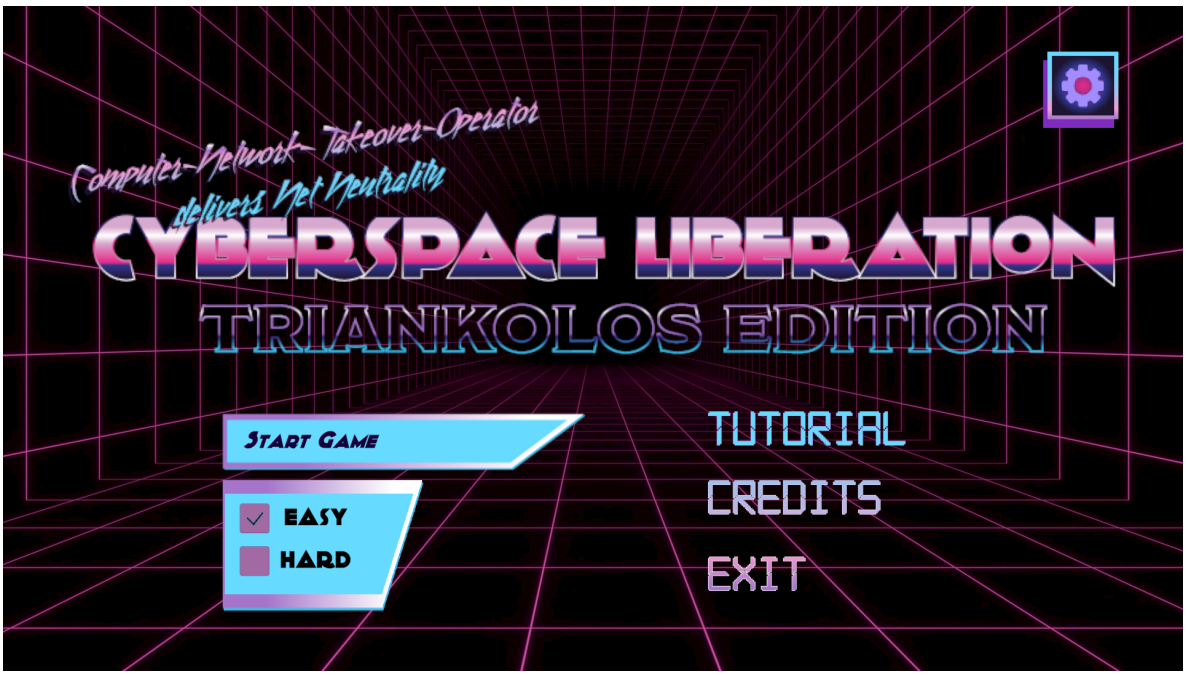


Fig. 5: Main Menu UI After

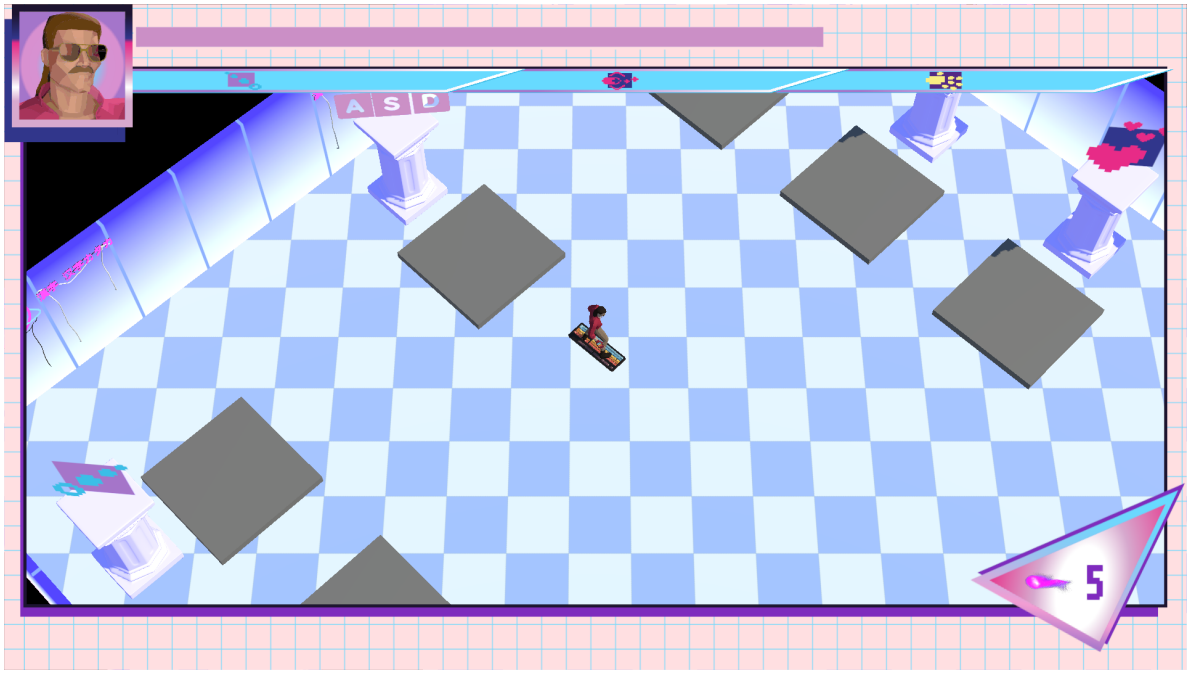


Fig. 6: In-Game UI Before

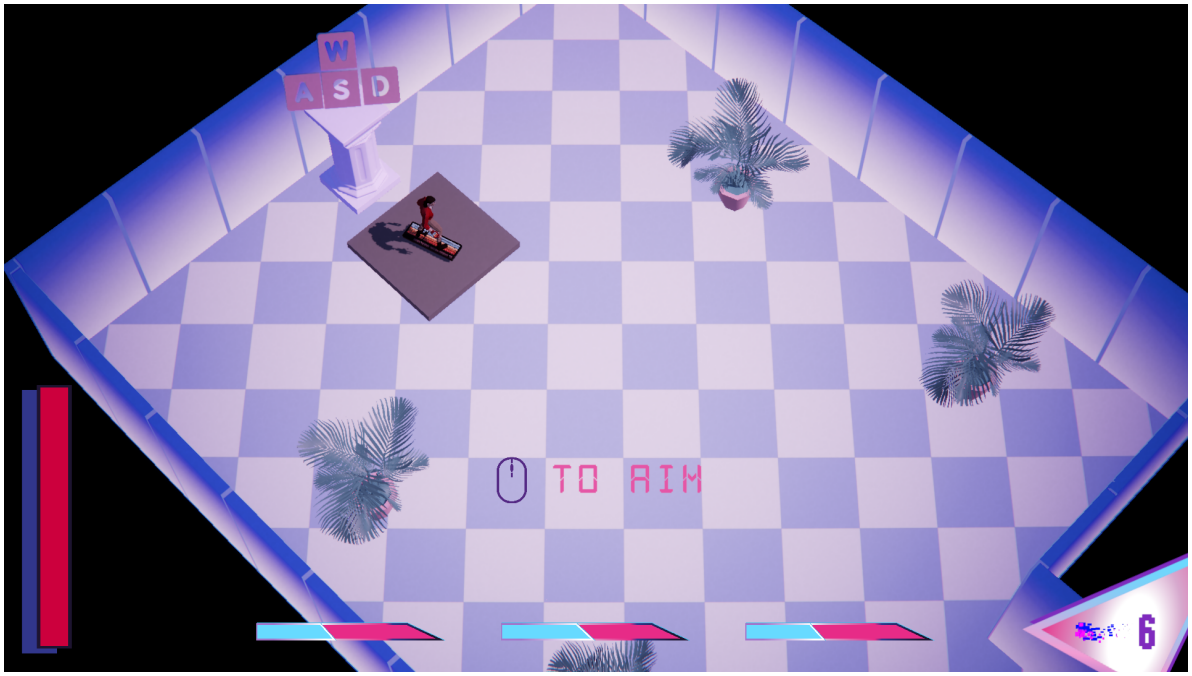


Fig. 7: In-Game UI After

Our Experience

While some of our initial design decisions made it into the actual game we did make many changes as we progressed along. At its core, our game is still a bullet hell/rogue-like with abilities. Our levels are being procedurally generated and we managed to implement two types of enemies as well as a boss. Our initial design also contained different weapons between which the player could choose. This feature got ultimately scratched. Based on feedback we added a progression system to the design, which we ultimately didn't have time to integrate.

We followed a development schedule, but not necessarily the one we developed during the first or second week of the course. We created the initial plan in Excel, but we felt that Excel is not particularly great as a project management tool, so we quickly switched to Trello. We then used our initial schedule together with the milestone deadlines to create a new, more in-depth, but short-term schedule, on a weekly or bi-weekly basis.

Our initial schedule was based on many assumptions about the game design, which features we thought were important, and how long they would take. We had no idea how long those features would actually take and many of our initial assumptions were revised anyway so our requirements changed frequently. We did manage to follow those shorter plans most of the time, however.

One issue with the project structure was that it doesn't leave too much room to iterate. Our initial design disregarded the theme to some degree and also was mechanically heavily derivative of similar games in the genre. Even now, with the final release coming up, we still weren't able to fix that problem with the only thing maybe setting the game apart being its visuals. With a less rigid project structure we maybe could have more easily gone back and fixed issues with the design, but due to the fixed structure we also had to prepare the deliverables for the next milestone so the problems we had just solidified.

Generally, working with a theme can be quite nice as it can provide a lot of guidance, especially if you are supposed to come up with a game design and don’t have any ideas in which direction to take it. It also sort of unifies all the games that are developed for the course which is nice. However, we felt that it wasn't communicated clearly enough how important the theme actually is. (could have asked tho) Initially, we were under the impression that the theme was meant to be merely guiding in finding an initial design. It should be used, but wouldn't be super important which is partly why we went for a bullet hell type of game. It would sort of check the theme, as bullet hell games tend to be Kind of chaotic, but would give us a lot of freedom in other regards. However, in the next three or so later presentations we received feedback that boiled down to: Theme should be more present and more important in your game.

However, everything we came up with to properly, deeply integrate the theme into our core mechanics would have required fundamental changes to the game. Time went by and we had deadlines for which we had to deliver and so the game progressed as well. With that it became less and less feasible to implement those changes as they would require huge amounts of reworks for which we simply did not have the time.

Towards the end of the project, we had a huge problem with motivation. We think in part this was because we felt we had to crunch to make the alpha release deadline. After that motivation was pretty low, but we still had to keep going as there were still four weeks until the end of the project.

Another issue was that we worked with a bunch of artists who had no experience working on games, so we had to manage that as well. We probably should have introduced them better as some of the issues we later experienced can be traced back to that.

The greatest success during the project was probably when we gave the game to players during playtesting. Our players all enjoyed the game and seeing people appreciate something you created is just a great feeling.

Overall, we are quite happy with the final result of our game, since players enjoyed it, which ultimately is the goal. However, we would not say that the game fully succeeded from our perspective. Personally, we all learned a lot during the development process. It was the first time we attempted procedural level generation and we are quite happy with the result. It was also the first time that we used behavior trees to implement an AI. So from a technical viewpoint, we would argue that we were successful. What we failed at, however, was

finding a clever design twist, in a time frame that would have allowed us to integrate it, to set us apart from similar games in the genre. As a result, our work is quite derivative and, at most, sets itself apart through its visuals. We are split on this last point, however. Some of us would have liked the design to be more unique. Some are fine with it, as we created a fairly polished game that fits well into a programmer's portfolio.

Overall, we enjoyed the course. We had a lot of fun making the game and we learned a lot doing it. We do have a few thoughts on how to improve the course, anyway. For one, we think that it might be good to iterate faster with shorter deadlines, but also shorter presentations. For example, it would have been nice to have less time for playtesting, then have a week or so to improve and fix the game and then have another week of playtesting to get feedback again and see if the game actually improved or whether new issues arose from the changes.

It would also be great if the course could adapt more to the needs of the individual games/groups. In our case, this could have meant more time to take a step back and fix issues with the initial design rather than having to somehow fix the initial design and at the same time press on, because the next deadline is breathing down our neck. We only have so much time for a course as this is not our only course after all.

We also think that it would be nice if some aspects of the course were more clearly communicated and less of a mess. For example, in some cases the deliverables that were required according to moodle differed from those listed in the project structure document. The project structure document felt kind of outdated in general which wasn't great. Another thing, that in hindsight would have been interesting to know, would be what we are being graded on. For example, we are still unsure how important the theme or a unique design actually are in the context of this course. We do need to admit that these communication failures are partly on us, however, as we always could have asked, but ultimately didn't.

As a final point, on which we are split, we would add that it could be worth thinking about how important design and how important tech should be in this course. While focussing on both can be immensely valuable, some students may be put off by this as this is a computer science degree after all, and some want to code rather than worry about design issues.