A TINY CRISIS IN SPACE

TUM - COMPUTER GAMES LABORATORY WINTER TERM 2017/2018

Robert Brand, Jean-Luc Etgen, Manuel Neuberger and Laura Vu

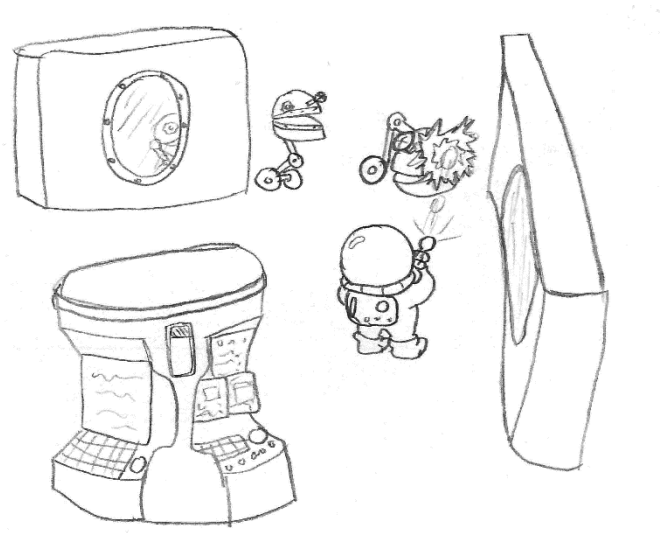
CONTENTS

Game Proposal 3
Game Description 3
Setting 3
Objectives and Game Structure 3
Mechanics / Player Actions 4
Enemies 5
Technical Achievement 6
Networking 6
Graphical Style 6
Big Idea Bullseye 6
Development Schedule 6
Layered Development Targets 6
Timeline 8
Assessment 8
Additional Concept Sketches 8
Paper Prototype 9
Design Goals 9
The modeled game 9
Materials 9
Game Rules – The players 10
Game Rules – The enemies 12
Conclusions 13
Observations 13
Critique 13
Progress Report 14
Functional Minimum 14
Low Target 15
Design Revisions 16

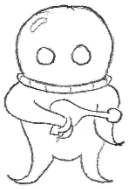
GAME PROPOSAL

GAME DESCRIPTION

"A Tiny Crisis in Space" is a Third-Person Shooter in which four players must cooperate to defend their bases against waves of AI-controlled enemies. To further incorporate the course theme, "Together", their main means of accomplishing this (besides shooting) is to merge with another player. This fused form is stronger, yet can only be in one place at a time and must be controlled together. Players therefore have to coordinate – not only to control the fused form but also to decide when to merge and when to (literally) split up.



SETTING

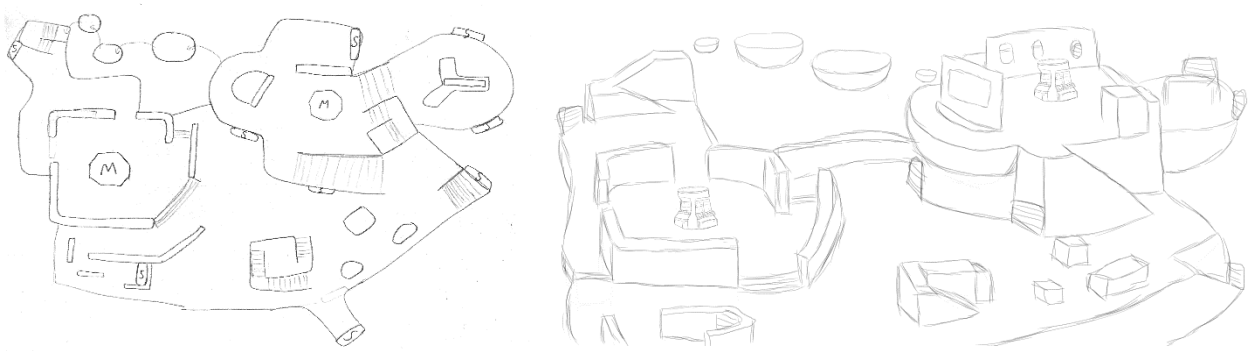


The players assume the roles of alien astronauts who are having a tiny bit of a crisis aboard their space station. The board computer has been infected by a malicious virus, causing it to go haywire and send out a multitude of robots to destroy the control panels. The players must defend these control panels while they perform an anti-virus scan to identify and eliminate the source of the board computer's malfunction.

OBJECTIVES AND GAME STRUCTURE

Within the game, the anti-virus scan translates to a timer. If the players can keep the enemies from destroying any control panels until the timer runs out, they win the game. Consequently, the game is lost if any of the control panels are destroyed. Additionally, enemies can also attack the players themselves, damaging and (if they are low enough) "downing" them. If all players are down at the same time, the game is also lost. To increase pressure, enemies will spawn more frequently and in greater number as the timer ticks down.

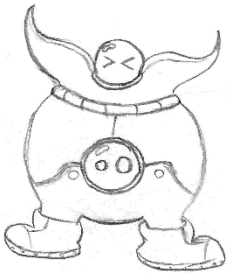
In some rounds, a secondary objective must be achieved in addition to protecting the control panels to win. Here, each panel requires three randomized robot parts to complete the scan. Each robot has a chance to drop a part specific to the robot's type upon defeat and the players must carry these back to the control panels. Only one such part can be carried at a time to discourage hoarding and encourage multiple players to cooperate in the part collection. This secondary objective is intended to get players to move away from the control panels instead of always staying close to them.



The space station's map design should include at least two control panels in different locations. This would require players to coordinate and split up to defend them both. This is necessary so that players cannot simply run around in their more powerful merged form all the time.

MECHANICS / PLAYER ACTIONS

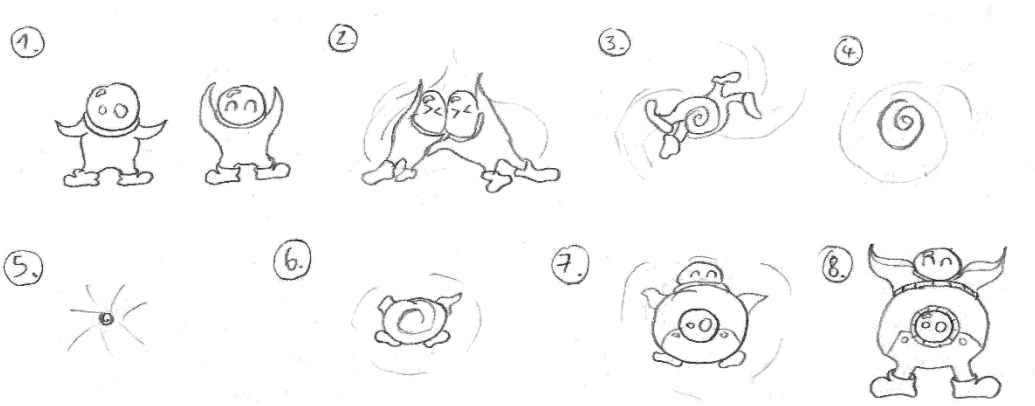
Individually, each player can shoot, run around and jump. If moving while shooting, the player will strafe, moving at a reduced speed. This discourages shooting permanently without completely disabling the player.



To merge, two players must stand within a small radius of each other and press a dedicated merge button. After the transformation, they will be in control of different parts of the same character. In this fused form, the players cannot shoot but instead throw powerful explosives. They must work together to adjust the trajectory. The bottom player controls the movement of the body while the top player can turn and aim independently. Both therefore have influence over where the explosives will land. Each player additionally controls one arm to throw with.

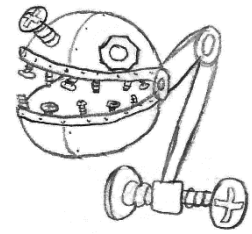
If a player is "downed" they can be revived by other players by standing beside them for a small duration. This again encourages cooperation between the players. It also prevents the game from becoming an experience where only one or two players remain and the others have to watch (which would very much go against the theme of "Together").

Lastly, even though players are expected to be in the same room or use a voice chat service while playing the game, they should be able to communicate within the game in some form. The players will thus have the ability to "ping" their location, making it visible as a beacon to their teammates.

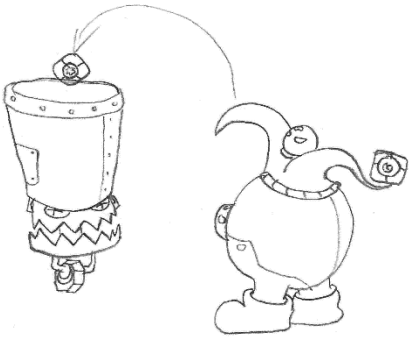


ENEMIES

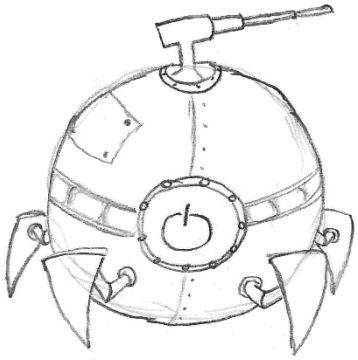
The robot enemies target a specific control panel when spawned. If a player comes within their range and they are not yet too close to the panel, they will instead start chasing the player. If the chased player is too far away they will re-target the closest panel. To start with, we intend to implement three different types of enemies.



The basic enemies should be the most common and easy to defeat. Individual players can defeat them by simply repeatedly shooting at them. In fused state, they are even easier to dispatch. However, they could still overwhelm players due to their numbers. Basic enemies can only attack at close range. To increase variety, these could come in varying sizes with different speeds and health points to match.

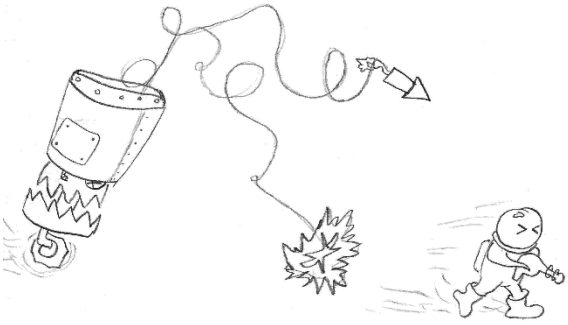


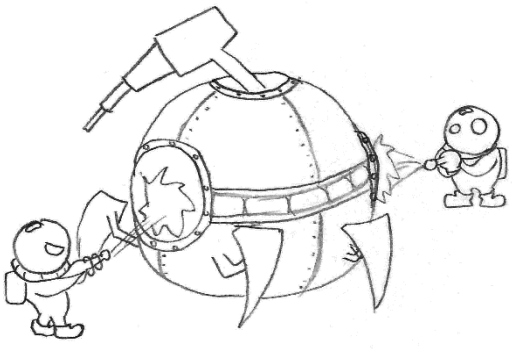
The first special enemy ("Robucket") can only be defeated by throwing an explosive into the opening in its head in fused form. This robot therefore requires players who are spread up to communicate and group up. It also poses a much more significant threat compared to basic enemies as it can shoot homing missiles at a medium range.



The second special enemy ("Orbot") can be considered the opposite of the first as it requires players to split up if fused. This is due to the fact that it can only be defeated by shooting the power buttons on its front and back at roughly the same time. To attack players or the control panels, this robot can shoot a laser beam from the cannon at its top.

These enemies encourage the use of both fused and individual forms.





TECHNICAL ACHIEVEMENT

We intend to implement the game using the Unity game engine. Unity provides a lot of functionality such as AI agent path finding and basic networking already. However, there is still a significant technical effort involved in adapting these features to our game. For example, though there is functionality for third person character movement provided, this has to be heavily customized to make for a suitable player controller for our game.

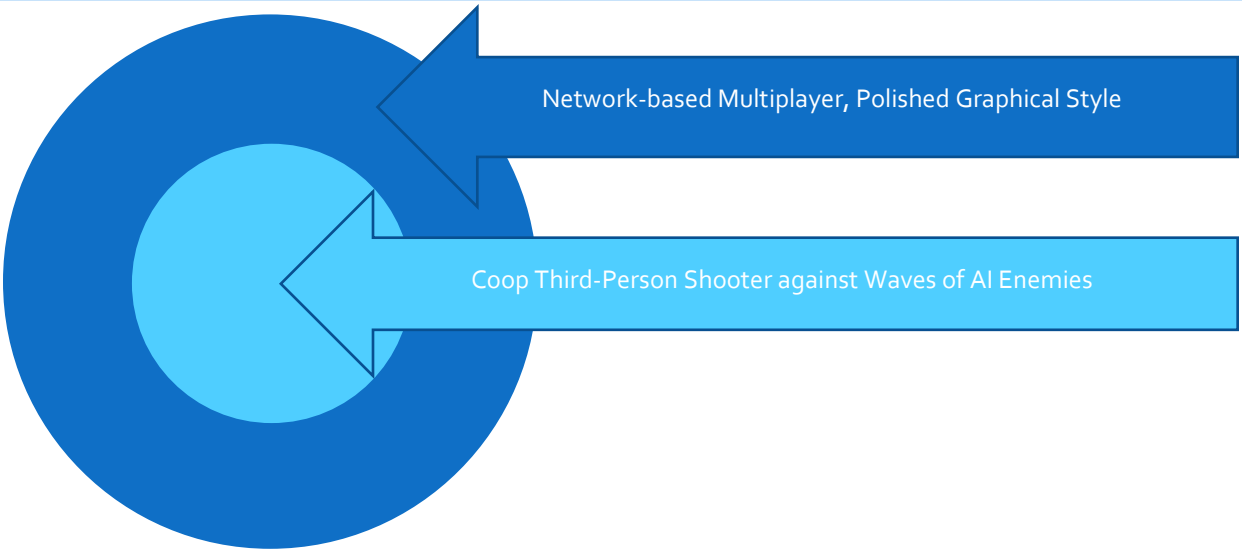
NETWORKING

Since the game is intended for four players with four different viewpoints, the best solution is to play at different PCs. Our primary technical challenge will therefore be to implement network-based multiplayer. There are several factors making this more difficult. Firstly, the game is going to be rather fast-paced. This means that the network code must ensure responsive, consistent behavior of all interactable elements within very short time frames. Secondly, the fused form will be challenging to implement since two inputs from different machines will be used to control the same character. As such, movement cannot only be handled client-side and sent to the server for verification.

GRAPHICAL STYLE

Our secondary focus will be to give the game a unique, polished look. Though much of this is more of an artistic challenge, there are certainly several technical challenges involved. For example, animations need to be synced to character movement to avoid issues such as a character's feet "sliding" on the ground. Custom shaders must also be implemented for some effects such as Rim Lighting.

BIG IDEA BULLSEYE



DEVELOPMENT SCHEDULE

LAYERED DEVELOPMENT TARGETS

Below the features that must be implemented for each layered target are specified. Note that each subsequent layer assumes that all functionality from previous layers has already been completed.

FUNCTIONAL MINIMUM

- Third Person Character Movement and Camera
- Shooting Mechanic and Third Person Aiming
- Enemies that Spawn, Move, somehow damage the Player and Objective and can be defeated
- Win and Lose Conditions (e.g. Player Health, Game Timer and Control Panel Health)
- A basic Level / Map (sort of a test level)

LOW TARGET

- Networking-based Multiplayer
- Merge Mechanic and Coordination-based Movement of the Merged Character
- 3D Art for Basic Enemy and Player Characters (Models, Animations, Textures)
- Basic 3D Environment Art
- Game Manager that spawns enemies in pre-determined waves
- An actual Map (i.e. where some thought went into the design, both visually and from a gameplay perspective)
- Basic sound effects
- Basic Game Menu

DESIRABLE TARGET

- Additional Enemies: 1 Enemy that can only be defeated in merged form, 1 Enemy that can only be defeated when positioned in two different spots
- Attack patterns for the new Enemies that make them different (i.e. 1 enemy that shoots a laser, 1 enemy that fires homing missiles)
- More Detailed Environment Art (Props)
- Visual Effects (that are not just placeholders) for Projectiles, Impacts, Explosions, etc.
- Ability to "ping" your current location for other players to see
- Custom Music (1 Background Music Track) + Improved SFX
- A more polished Game Menu

HIGH TARGET

- Secondary Goal (Enemies drop parts that must be carried towards the correct Control Panel)
- Objects which make the map more interesting: Trampolines, Zip-Lines, See-Saw Catapult, etc.
- Improved Game Manager that spawns Enemy Waves in a randomized way depending on a difficulty level and game timer
- More Background Music

EXTRAS

- Multiple Merges: Depending on which players merge, the resulting fusion is different (different way of moving, different weapon)
- Corresponding Enemies that require specific merge-abilities to defeat
- Multiple Maps

- Events that sometimes change the way a wave plays out: E.g. Lowered Gravity, Enemy frenzy (faster movement), A special Boss Enemy which requires all players to coordinate more than normal to defeat, Moving Control Panels, etc.
- Matchmaking / Lobbies

TIMELINE

See Attachment. Colors correspond to the team member (primarily) assigned to the task.

Everyone Robert Jean-Luc Manuel Laura

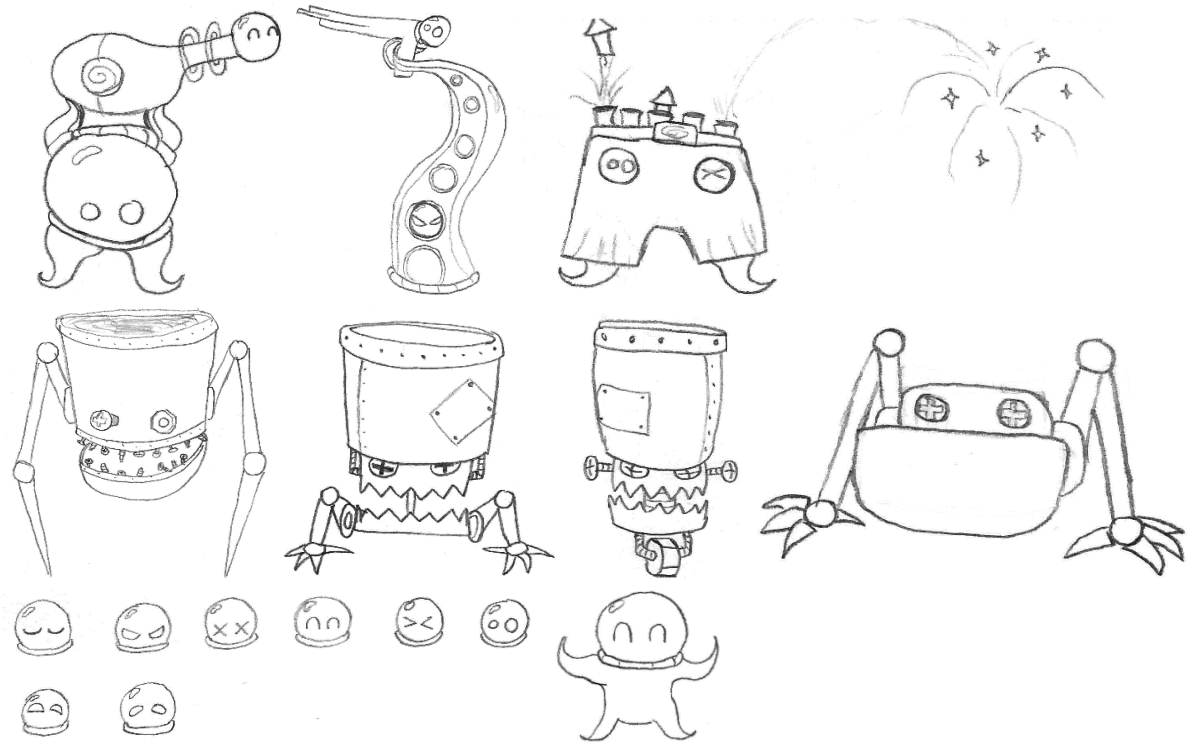
ASSESSMENT

"A Tiny Crisis in Space" is intended as a game which a group of friends plays (preferably while somehow able to simultaneously talk to one another). Even though the shooting gameplay itself should be fun, what sets this game apart and makes it cool is the requirement to work together and coordinate. The players should feel a sense of teamwork as they purposefully split up to cover different parts of the map or literally combine their abilities to take down immediate threats.

Our design goal is therefore to create situations in the game which require coordination, communication and teamwork. It is also important that these situations can neither be solved by always sticking together nor by every player "doing their own thing" but by alternating between the two.

Due to a comic-like aesthetic the game should be playable by all ages. However, a certain level of dexterity and quick thinking will be needed in addition to the aforementioned coordination skills to succeed at the game.

ADDITIONAL CONCEPT SKETCHES



PAPER PROTOTYPE

DESIGN GOALS

To model the game "A Tiny Crisis in Space", a paper prototype was built. This was done to make the game playable in a very early stage of the development process. So, design problems can be highlighted quickly and be avoided.

The core mechanic of "A Tiny Crisis in Space" is the possibility to merge with other players. To make this relevant in the game the player must have a reason to merge and demerge. In other words, there must be at least one situation where the other form is so much stronger that it is worth to merge.

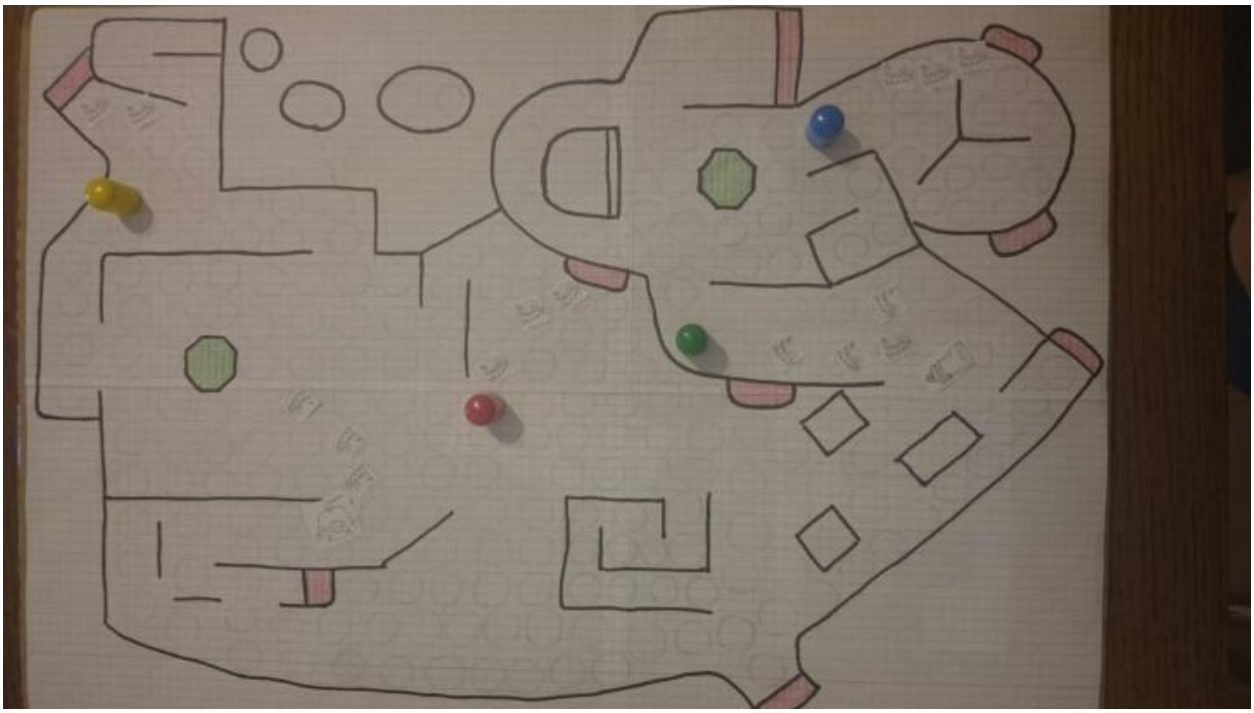
In the game, these situations will be caused by different enemy types. The unique weaknesses of the "Robucket" and the "Orbot" shall ensure that the players merge and demerge frequently.

Our goal with this prototype was therefore to investigate this meta-gameplay of enemies spawning and causing players to have to alternate between the two states. The prototype is not intended to investigate the primary gameplay of shooting, jumping and general player movement as these elements would be very hard to represent in a paper version.

THE MODELED GAME

MATERIALS

A map was designed to contain all imported elements which will also be found in the final game. These include enemy spawn points which are represented by the red areas in the picture below, as well as the control panels which are marked green. The four players are represented by tokens taken from other games while the enemies are modeled by little cards.



While in the final game the players will be able to move freely here it was modeled into discrete fields, which can be seen as circles in the picture.

GAME RULES – THE PLAYERS

While in the final game everything will happen in parallel, for the board game it was decided to divide the game into player and enemy turns.

Because the players work together anyway and they will be encouraged to use a voice chat in the final game, their turns are parallel to save time. Also, they can discuss their actions with each other. Each player has 1 HP while a base has 3 HP. If the base falls to zero HP, the game is lost. If a player falls to zero HP, he is knocked out and must be revived by another player.

Each turn every player can decide between different actions. These differ depending on if the players are currently merged.

Single:

While a player is not merged he or she can decide between four actions: Attack, Run, Merge, Revive. Except for the merge-action, only one action can be performed each turn.

Attack:

The player may attack one enemy within a range of 5 fields or less. What happens to the enemy afterward depends on the enemy type. Also, the player may move 1 field or jump the same distance. Jumping is only possible on the platforms in the left corner of the map or to jump down from a high ground.

Run:

The player may move 4 fields or less. Each of these movements may also be a jump. Jumping is only possible on the platforms in the left corner of the map or to jump down from a high ground.

Merge:

If two players stand next to each other (this may be directly or diagonal), they may merge into one player with a different action set. This is only possible if both players agree.

This is the only action which does not cost a full turn. It is possible to merge and still do other actions afterward, but not the other way around. While in the final game merging will, of course, take some time, for the model the loss in time makes the merging possibility less interesting.

Revive:

If a player stands next to a knocked-out player he or she may revive him/her. The revived player may immediately perform an action.

Merged form:

After two players have merged it will be decided randomly which player moves and which player shoots. So, each turns the players only have the possibility to do their action (move or shot), revive or demerge. Which of the players does his action first is up to them.

Attack:

The player responsible for shooting may throw a grenade to any field which is either two or three fields away from the player. This grenade destroys all basic enemies on and around this field. A "Robucket" is destroyed if the grenade lands on his field, but then no basic enemies on the surrounding fields are destroyed.

Move:

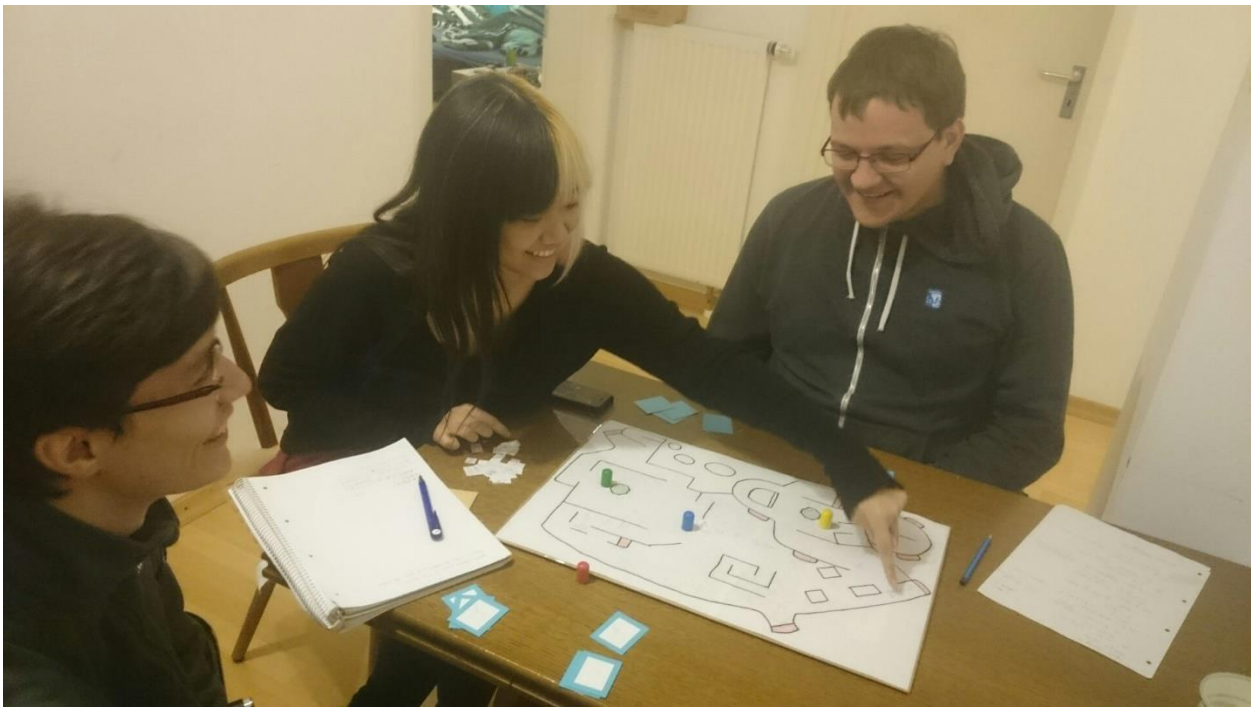
The player responsible for walking may move up to 2 fields in any direction. This movement may also be a jump.

Demerge:

Like with merging if both players agree they can separate. Players may demerge after every action they have performed, but are not allowed to perform their action afterward.

Revive:

Either player may sacrifice his/her turn to revive a knocked-out player.



GAME RULES – THE ENEMIES

The enemies are controlled by an additional player who simulates the pc. At predefined rounds, waves of enemies spawn at the spawn points. Depending on the enemy type the move or attack in certain situations.

Basic Enemy:

The basic Enemy moves two fields per round to the closest base if there is no player in a two-field range. If there is a player or a base in the two-field range the enemy will attack by walking next to the target and then causing one point of damage.

If it is hit by one bullet or in the explosion range of a grenade it will be removed from the field.

Orbot Enemy:

The "Orbot" Enemy moves two fields per round to the closest base if there is no player in a six-field range.

If there is a target in this range, it will charge a laser. When charging it cannot move but may move the next round when the laser is fired. When the laser is fired it causes 2 damage to all players or bases in a straight line of a six-field range. The line starts at the closest target.

When the "Orbot" is shot by 2 single players in the same turn on opposite sides, it is destroyed. It is immune to grenades.

Robucket Enemy:

The "Robucket" Enemy moves two fields per round to the closest base if there is no player in a three-field range. If there is a target in this range, it will shoot missiles at the field where the target is standing which will land one round later. Upon impact, all targets which stand on the field or next to it will receive two damage.

When a grenade is fired onto the field where the "Robucket" stands it will be destroyed.

CONCLUSIONS

OBSERVATIONS

While playing the game important conclusions were obtained. Some problems like a difficult to handle enemy AI by hand occurred. But as these problems are restricted to the paper prototype there will not be any big focus on solving them. Besides this the game was enjoyable to play, especially tactical planning and strategies were very entertaining. Through the steadily changing enemy distribution, players tended to merge and demerge quite often.

It was also observed that for an interesting game, which encourages the merging and demerging the design of the map is of great importance. Enemy spawners that are too close to a control panel restrict the players from moving away from this panel and therefore should be removed. Also, shortcuts to the players are very imported, because otherwise, they do not have the possibility to react to enemies which are too far away.

CRITIQUE

While the paper prototype provided a great possibility to verify the core game element, many concessions had to be made.

While in the prototype the player had a complete overview of the map, in the final game they will not have it. Furthermore, players in the final game should be more stressed than in the paper prototype due to quick decisions, which could not be modeled well here. Also for the prototype, a perfect aiming of all players was assumed which is not realistic for the final game.

PROGRESS REPORT

At this stage in the game project our goal was to implement all features that are part of the *Functional Minimum* and *Low Target*.

FUNCTIONAL MINIMUM

We have implemented all the features listed in *Functional Minimum*. For the third-person character movement and camera the Unity Standard asset was used and some of the code was repurposed to handle strafing. Additionally, air jumping and air control had to be added. Since the controller was built using the root motion of the walk animation of the character we decided to include that too in our character's animations.

Since we are using a non-parented third person camera it must find the player in the scene. The problem is that there are multiple players with the "Player" tag and it should find the player who is locally playing. This was also fixed with some tweaks to the camera code.

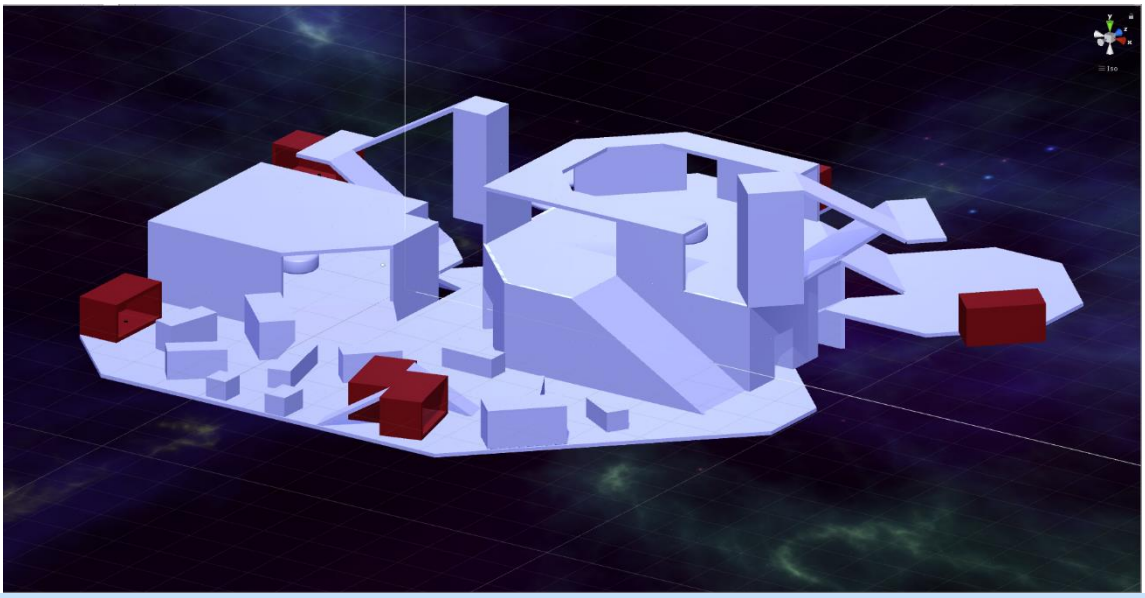
Functionally, the shooting and aiming mechanic was fully implemented, though there are some problems that still need to be addressed. The downside of aiming in third person is that the point the player is aiming the reticule has some offset to where the bullet starts on the player's gun. This leads to visibility problems as the player thinks he can shoot something just to find out that the bullet's path is obstructed. The way this could be fixed is that the bullet moves slightly upwards towards the ray that is shot from the camera before moving along it.

The basic fodder enemy is currently almost fully implemented. Only the death animation and color texture are missing. Enemies currently used the Unity *NavMesh* pathfinding system to find objectives and players. The integration went well and enemies use the shortest path to the nearest objective when they spawn. If they see a player in range they will try to attack the player until he dies, unless the objective is closer.

The high-level game logic for winning and losing is handled by the *GameStateManager*, which exists client- and server-side. The server-side script checks that enough players have joined and then starts the game. It also checks for the players' HP and bases' HP and triggers the loss state if all players are dead or one of the bases is destroyed. When the wave is over and none of the loss conditions occurred, the players win. All states and timings are synchronized to clients so even if a player disconnects during the game he can still rejoin and continue playing.

Our map that is currently in the build is basically a 3D representation of the map shown in our paper prototype. The southern area was totally reworked as it felt empty so we placed containers there to make it more interesting. The western base is now enclosed with a ceiling to make it harder for players to see everything when they are at the top base. A spawnpoint was added in the middle between the 2 bases with a tunnel connecting the eastern part of the map with the western part of the map. Because the 2D map of the paper prototype can't show the verticality of the

level we added that with the 3D version. Although the map is functionally done and could be used in the final game, it will require a lot of balancing using the enemy spawn manager.



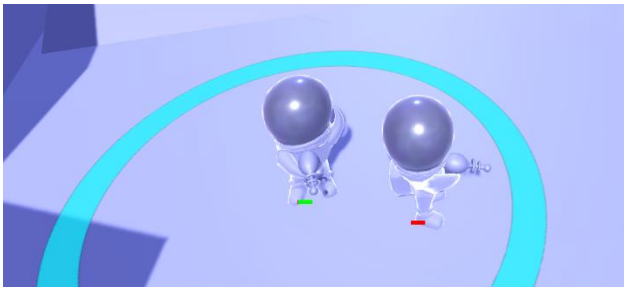
LOW TARGET

Most of the goals for the *Low Target* have been fulfilled in the current build. Due to time constraints, the implementation of sound effects and the game menu have been postponed.

Unity Networking comes with a few difficulties for beginners. As soon as we understood the basic concepts of networking coding with remote procedure calls, commands and synchronized variables all further implementation went much faster. Although transform synchronization works well with good ping and no packet loss, animations synchronization is much harder. Due to the way Unity's default implementation of *NetworkAnimators* works, some animations are just too choppy and very noticeably worse on a client than on the server. If we feel that this becomes worse over the course of development we will create our own implementation of network animation.

Health is also an important aspect of our game. The way it is currently implemented not just the server can change the health of enemies, players and bases but also clients. This is of course unsafe as a malicious client could be sending false information but we just hope that never happens. The advantage this brings is that the death zone at the bottom of the map where players can fall to and reviving other players was much easier to implement.

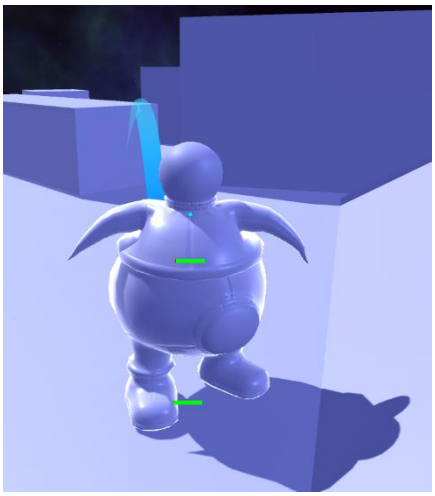
Most of the player's animations are already present in our game. Players wave their hand when they want to merge, walking and strafing animation blend trees exist and jumping is supported by the animator. Things that are still missing for the *High Target* are color textures and a "downed" animation when the player needs to be revived.



An enemy spawn manager uses the *Wave* data structure to spawn enemies in the level. A wave is defined by the time players need to survive to win and a list of enemy groups. An enemy group has a spawn time and a set of prefabs. The way our spawning system handles groups is by assigning it to one of the pre-defined spawn points in the level, which then cannot spawn another group for a certain time. In this way, we ensure that our enemies always

spawn differently every time but also distribute around the map. Currently, only one wave is supported but in future implementations we are aiming to have 3 waves of varying difficulties which should last 3 minutes survival time each.

DESIGN REVISIONS



Currently just one merge combination is possible: the upper player can throw bombs with 2 hands and the lower player control movement and jumping. This is something we changed from our previous design where both players could throw grenades. We felt that the lower player might not have enough to do when in the merged state but this turned out not to be the case in our current implementation so we left it that way. The upper player has 2 scripts regarding the merge: one only sends its rotation about the local y axis to the server and one looks for the position of the lower player and move to a set position above the lower player on his client. Because we wanted to sync animations on both sides we use the *NetworkAnimator* to animate both player objects and we use the rotation script of the upper player to correctly align the upper player.

We also made some minor additions to existing features that we had not thought of when we designed them. Player cannot merge when they are low health and downed otherwise they could merge with full health. Furthermore, when player are low health in their merged form they are forcefully demerged. Another feature we implemented was that the player that requests the merge first always becomes the lower player who controls movement. The player that "joins" the merge second will always be the upper player.

This will tie in to a new feature we would like to implement called demerge jumping. When both player want to demerge, the upper player should be able to catapult himself upwards, higher than a normal jump. This should give players more mobility than enemies and give the more incentives to stay together other than the advantages the merge itself brings.