TUM Travel:
The Final Chance
Milestone IV

Milestone I

1. GAME DESCRIPTION

GAME LORE: "A university student who just graduated walks into the MI building for the last time as a registered student. As he walks down the magistrale, he feels both nostalgic and proud at the same time. Once he is in front of the library, he spots a weird anomaly right next to the lockers at the library entrance. The anomaly he spotted was nothing more than a portal that opened into a computer laboratory. Apparently, the LRZ computer lab at Garching was experimenting with a time-space continuum that caused unexpected behaviours around the campus for the first time. Unluckily, our protagonist was the first person to witness such a thing. As he approaches the portal, he gets sucked into it and finds himself in front of the SuperMUC-NG, LRZ's supercomputer, which is now capable of sending TUM students back two years into the past. After the reality around him begins to shake unexpectedly, he doesn't understand what's going on and checks his phone to see if everything progresses in a normal manner. That is when he realizes that he has travelled in time back to semester one. Although the machine's capabilities were unknown so far, he was the first to experience these ambiguities in time. Then, a brilliant idea pops into his head. This time, all the frustration and suffering he experienced during his university years have now created a unique opportunity for him to relive these experiences; everything will happen the way he wants. He has a chance to take revenge on all of the painful courses that caused him anxiety and sleepless nights with one goal in mind. To beat the students who caused him to get curved down."

The game starts with the aforesaid story in mind. After a short cutscene to establish the character's presence, we decided to include a brief intro animation similar to the beginning intro of the famous game Pizza Tower (link). Then, the student finds himself in the Magistrale corridor, which corresponds to the very beginning of the game.

Design Decisions Grounded in Theme:

- Time travel & Nostalgia + redo life choices = *literal connection to the theme*
- Academic struggle & performance & familiar environment = *emotional connection to the theme*
- Portal anomaly = *sci-fi / surreal tone*
- Comic satire = distancing from reality = *thematic intent, captures the literal and emotional connection*

Game Genre&Design Choices: The game belongs to the classic **Beat'em Up** genre, where the player has to beat waves of enemies, which are our beloved students, TAs, and Professors, with a clear intention of portraying everyone we'll include in a funny, friendly, and anonymous manner. We respect everyone equally and definitely don't want to offend anyone while making this game. We thought that having a comic-style design, reminiscent of 90s drawings, would best suit this game and genre, as it creates a retro feel that complements the time-traveling theme in TUM. Since there are more than enough creative processes planned for this game (and considering we are just a team of two students with limited time), we thought the game shouldn't contain more than 3 levels. Although these levels are not specifically designed yet, they will be designed following the ideas in mind: one wave-focused area in which the foes will be thrown at the player (a brawling style), one special level that incorporates a pre-boss fight phase with TAs, and lastly, one boss fight with specific Professor/Professors to defeat as a final fight. Scenes will take place in a 2.5D environment, scrolling in a belt-like manner, with depth (tilted z-axis) also included in the movable parts of the environment. We believe that restricting movement to a 2D plane may reduce the player's freedom when interacting with foes.

We aim to have fluent gameplay that doesn't feel dull and disconnected. To achieve this, we planned on having adaptive music that intensifies as the player performs better and enhances the gameplay experience. Composing high-BPM songs with FL Studio and FMOD will be one of our two primary goals, along with developing a highly responsive combat system. We believe that these two processes must be developed concurrently to achieve maximum coherence in the gameplay. To increase and decrease the tension during the fighting sequences, we prefer integrating vertical reorchestration by adding/removing musical instruments such as beats, snares, and melodies. This method, together with track blending (horizontal resequencing), will enable us to create seamless transitions that reflect the current feeling of the stage the player is in. Additionally, various combos will feature sound effects that are somewhat compatible with the track in the background. Although this might be hard to achieve in the first place, we might just have sound effects that don't disrupt the music in the background. On the other hand, onomatopoeia effects from comics will be incorporated into the character's combos to enhance responsiveness in their actions.

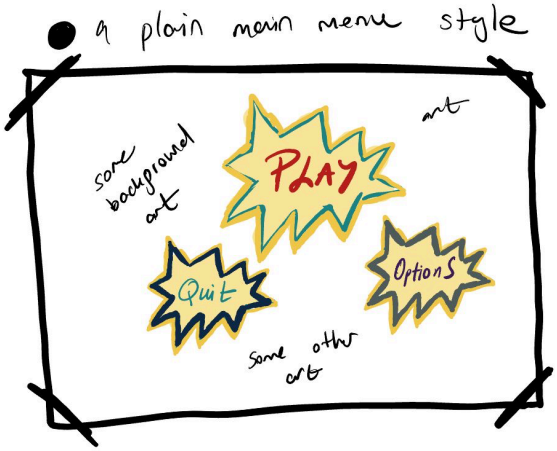
During the brainstorming phase, we thought that adding roguelite elements, which introduce meta progression into the game, would be compatible with the current theme we have. However, this requires designing complex combinations, arbitrary items that contribute to the permanent progression of the player in a run, or intricate skill trees. Even though these properties would add big depth to our game, we find it challenging to develop them, considering the time we have ahead. However, the good news is these are still on the table, just not as a goal for this semester's project. Instead, we want to integrate a simple checkpoint system where dying regresses the player's progression by moving the player back to the beginning of the current level, which is considerably smaller compared to the previous idea we had in mind. Our team considered removing this game attribute, which became a final decision after reviewing the critiques on the gameslab's wiki page. Currently, we plan to create at least three different fighting kits to introduce various attack variants to the gameplay. By fighting kits, we mean various tools to fight with foes, in addition to basic kicking and punching animations. We can't overlook the fact

that hand drawings take a lot of time and revision. Therefore, we might consider rigging the character versions instead of drawing each animation frame by hand. This way, we'll save time and focus on other more important aspects of the game.

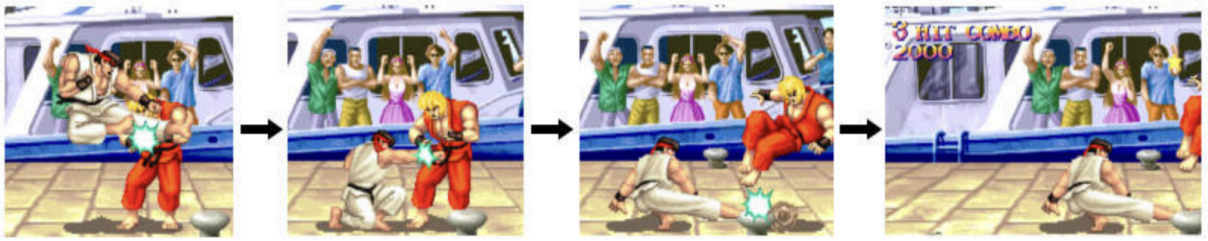
Game's "Target":

- *Tone: energetic, self-aware, humorous academic struggle fantasy.*
- *Feel: crunchy, rhythmic, readable, satisfying.*
- *Goal: not punishing rhythm game — rewarding flow state.*

Main Menu sketch:

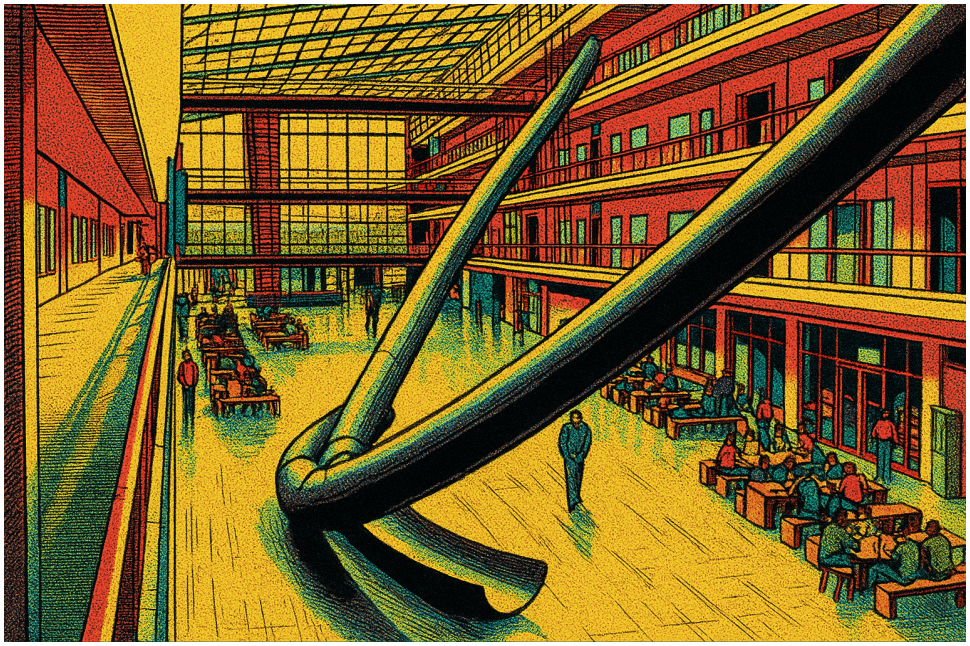


Player Mechanics:



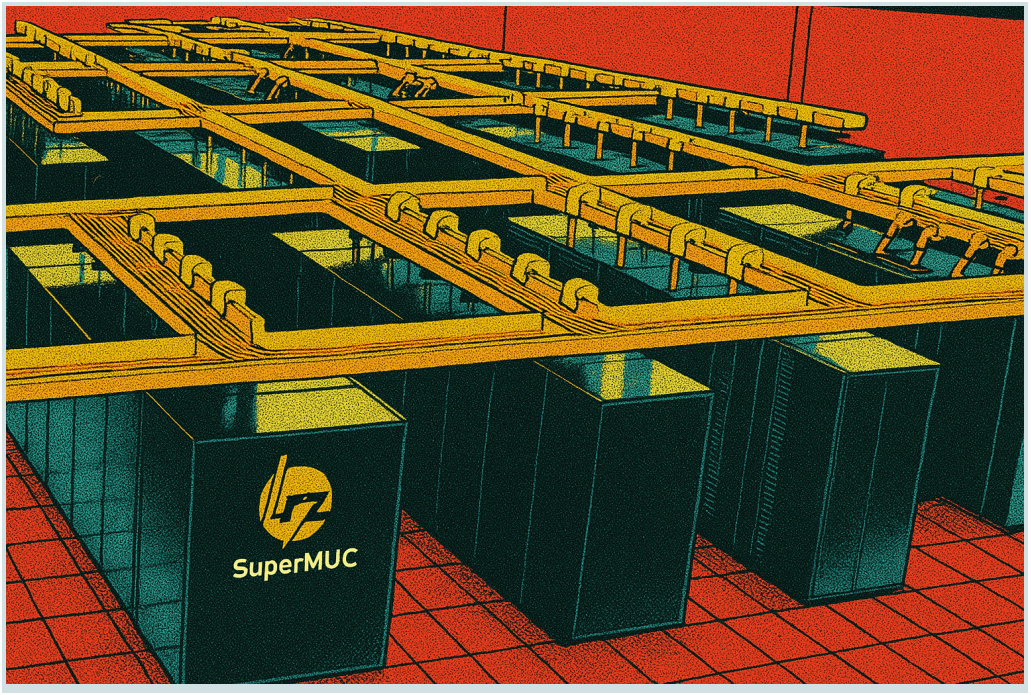
[Jumping HK -> crouching MP] is one part, and [Crouching MP -> crouching HK] is another. Combine them for a 3-hit combo.

Level 1: Magistrale — tutorial waves and brawling



Level 2: MI Bibliothek— pre-boss phase

Level 3: LRZ core — professor boss arena



TECHNICAL CONSIDERATIONS: We considered using Unity as our game engine, taking into account our experience and the game's design. Unity's URP rendering pipeline would best suit our needs, as we don't require photo-realism in our game, but rather expect a VFX-heavy experience. For this, Unity's Particle System (Shuriken), Unity's VFX Graph (URP compatible), and Unity's Shader Graph will be in our tech stack for implementing the game feeling mentioned in the earlier sections.

The game is initially planned to be released on Windows and Mac as the target platform choices. On both platforms, the game will be playable with either a controller or a keyboard. As version control software, we prefer Git with Git LFS for tracking large game assets.

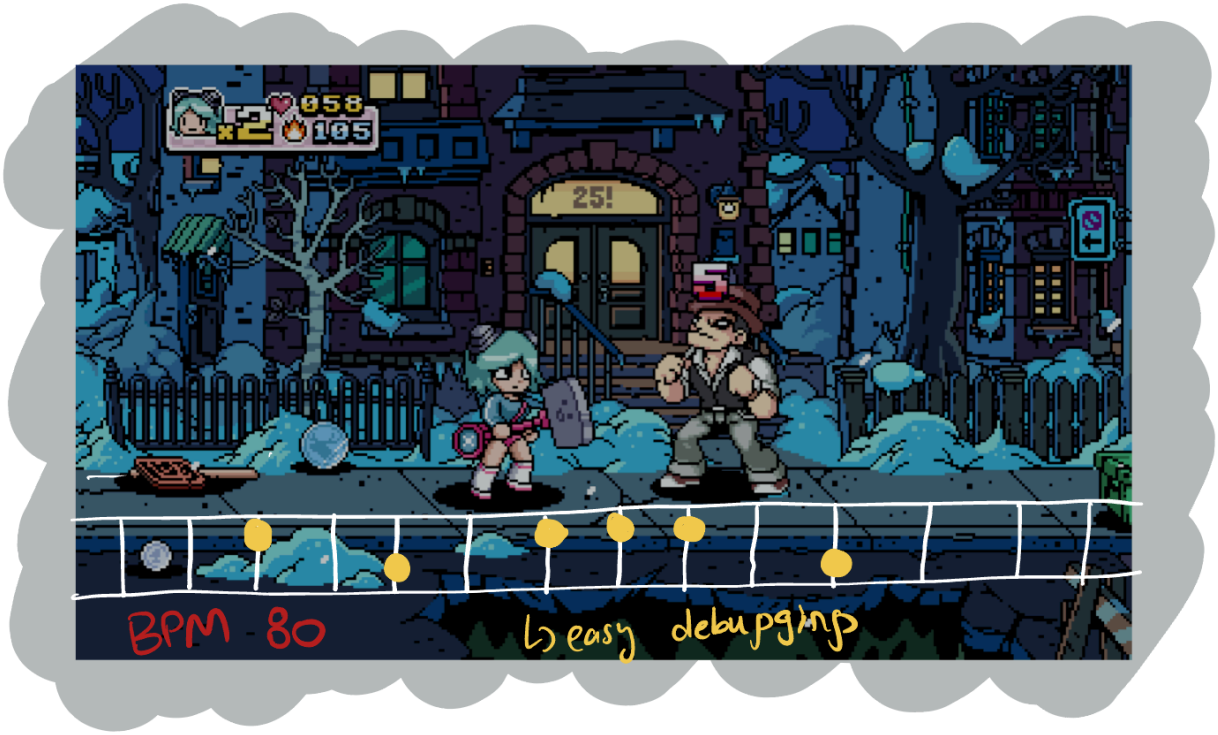
To us, one of the most important technical considerations is the NPC behaviour. Using a Hierarchical Finite State Machine for determining states, as well as using a Utility AI for picking the actions, would be the better choice for the scope of this game. Since we don't want the transitions to be perceived as too strict or dull, sub-state management within the combat would make the NPCs appear more intelligent. Additionally, we are aware of the problems, such as NPC attack coordination, positioning, and selecting the appropriate action in a given situation when multiple NPCs are present. To control and balance this, we decided to use an Enemy Group Coordinator (e.g., one attacker, one flanker, others circle/taunt) to help the player get some breathing room during the fight

sequences. For performance reasons, staggering enemy AI updates and tick throttling are great technical design choices for scenes where a bunch of enemies are thrown at the player in waves.

Saving the player's progress and spawning them at a checkpoint after death should be considered. Saving checkpoints to a JSON file or using Unity's Player Prefs would enable us to save progress via completed actions or passed stages in the game. This method is both cheap and easy to implement with a Save Manager script.

Mock UI / UX and HUD Vision:

Optional BPM HUD (debug + overlay):



Comic onomatopoeia pop-ups:

GPA Score Progression:

3.4 → 3.4 → 3.0

FUTURE PLANS&DLC CONTENT: Since the game in our minds is hard enough to implement in a coherent way, plans in this section are for future considerations. Previously, we mentioned having adaptive music incorporated into our game. To extend this, we thought of not only having an adaptive music scheme but also having this music in sync with the gameplay. The player's movement, combos, and interactions with the foes would also be synced with the track's BPM. In short, having a similar gameplay to HiFi Rush would be the ideal scenario. That is why this will be implemented if we have enough time after stabilizing all of the stages we mentioned in the above sections.
Additionally, we plan to add a local co-op feature to the game mode. We believe this immensely expands the gameplay experience of a beat'em up game, since having a partner in crime in this type of game is always more fun.

2. TECHNICAL ACHIEVEMENT

The core technical focus of our project is the development of a polished experience which involves fighting smart NPCs and having a performance-aware adaptive music engine. While our game is not primarily a rhythm game, we aim to meaningfully merge musical structure with beat'em up gameplay to enhance responsiveness, player immersion, and combat feedback. As mentioned before, we will consider using FMOD to achieve this within Unity.

The technical goal can be further analyzed in the following:

- The soundtrack dynamically reacts to combat intensity (vertical re-orchestration & horizontal re-sequencing) - **MAIN GOAL**
- Visual feedback (VFX pulses, hit flashes, camera bumps) enhance gameplay experience - **MAIN GOAL**
- Player attacks and hit reactions selectively align with BPM-based timing - **NICE TO HAVE**
- AI attack patterns are subtly biased towards the beat accents to create perceived flow without forcing rhythm play - **NICE TO HAVE**

To achieve this, we will:

- Follow basic game design principles for handling NPC states
- Use FMOD timeline beat callbacks
- Trigger adaptive audio layers based on gameplay parameters (combo state, wave intensity, hit streaks)
- DSP clock synchronization

3. BULLSEYE

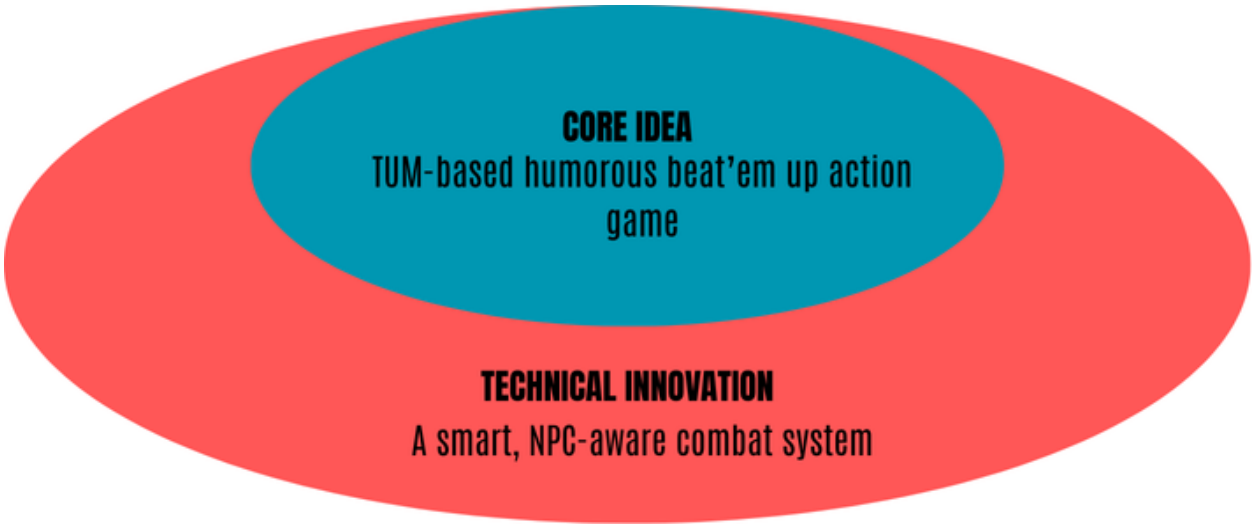
The big idea behind our game is to create an experience that all players will enjoy and find creative, but also an experience that TUM students, especially, will appreciate, as the whole narrative and locations within the game revolve around themes and areas that every person studying at this university has experienced.

At its core, our project is a humorous game that relives academic life at TUM and reshapes it through exaggerated combat and playful time-loop storytelling. The game's world, characters, humor, and tone build on recognizable experiences from the TUM student journey, turned into a comic and energetic beat 'em up.

Technically, the game innovates by integrating an **adaptive music system alongside engaging NPC encounters**. The two elements — the **TUM time-loop comedic vision** and the **smart combat system** — represent the central focus of our project and guide all design and implementation decisions.

CORE IDEA (Inner Circle):
Relive your studying journey by fighting through exaggerated academic challenges in a comic-style, time-loop beat 'em up — where student struggle becomes the motive to beat up your favourite TUM associates.

Technical Innovation (Outer Circle):
A smart, NPC-aware combat system that synchronizes audio, combat flow, visual feedback, and responsive gameplay cues.



4. DEVELOPMENT SCHEDULE

Functional Minimum:

- One composed music track, playing in the background
- One playable level with a single enemy archetype
- Basic enemy AI using a **simple FSM** (idle → chase/attack → retreat/defend)
- Keyboard + controller input support
- Minimal placeholder art (rough hand-drawn sketches/prototypes)
- Basic hit feedback (e.g., flicker, sound, simple particles)

Low Target:

- One composed track with **clear adaptive layers**
- Two playable levels with **at least two enemy types**
- **Improved FSM** with sub-states (movement, attack variants, defensive behavior)
- Basic visual feedback synchronized with the music (impact flashes, screen shake triggered on beat accents)
- Simple UI (health bar, score/combo)
- First pass on rigged/hand-drawn animations + improved character art direction

Desirable Target:

- Two composed adaptive tracks, each tied to their level's pacing and player performance
- Three playable levels, including **one boss encounter with a simple phase change**
- **HFSM + Utility AI** for enemy behavior (sub-state action selection, varied attacks)
- Fully animated main character and enemy animations
- Refined hit responses (hitstop, camera shake, onomatopoeia pop-ups)
- Improved environment art representing recognizable TUM spaces

High Target:

- Adaptive soundtrack for **all three levels**, plus boss theme
- **Boss with rhythmic-phase mechanics** and multiple attack behaviors
- Further-refined combat readability and polish (anticipation frames, stagger logic, enemy roles)
- **BPM/Beat HUD** for music/feedback debugging
- Additional environment details and unique assets capturing TUM identity
- Optional cooperative testing groundwork (not full multiplayer)

Extras:

- Public **Steam page** and playable Steam demo
- Entire game synchronized to beat timing (Hi-Fi Rush-style precision rhythm integration)
- More enemy archetypes, boss encounters, and student-life environments
- Expanded meta progression or lightweight roguelite upgrade system
- Local co-op mode with shared rhythm systems
- Full soundtrack album release

Task	Category	Start Date	End Date	Assignee
Proposal Report Prep	ORG	11/1/2025	11/5/2025	Both
Initial Sketches & Style Tests	ART	11/1/2025	11/7/2025	Nick
Track 1 Composition	AUDIO	11/1/2025	11/8/2025	Berke
Core Player Controller	DEV	11/1/2025	11/10/2025	Berke
FMOD Integration (Unity)	DEV	11/3/2025	11/10/2025	Nick
Adaptive Music Prototype	AUDIO	11/3/2025	11/17/2025	Nick
First Enemy Design	ART	11/5/2025	11/12/2025	Nick
Simple FSM Enemy	DEV	11/6/2025	11/13/2025	Berke
Prototype Packaging	ORG	11/8/2025	11/12/2025	Both
UI + HUD Implementation	DEV	11/10/2025	11/17/2025	Nick
Rig & Animate Enemy 1	ART	11/10/2025	11/18/2025	Nick
FMOD Adaptive Structure	AUDIO	11/10/2025	11/20/2025	Berke
Environment Art — Level 1	ART	11/8/2025	11/20/2025	Nick
Second Enemy + Miniboss Design	ART	11/15/2025	11/27/2025	Nick
Musical Effects for Hits	AUDIO	11/20/2025	12/5/2025	Berke
Enemy Utility AI + Coordinator	DEV	11/20/2025	12/5/2025	Berke
Level 1 Build	DEV	11/18/2025	11/27/2025	Nick
Rig & Animate Enemy 2 + Miniboss	ART	11/22/2025	12/3/2025	Nick
Environment Art — Level 2	ART	11/21/2025	12/4/2025	Nick
Track 2 Composition	AUDIO	11/25/2025	12/5/2025	Berke
Level 2 Build	DEV	11/28/2025	12/10/2025	Berke
FX Sprites (Onomatopoeia)	ART	12/1/2025	12/8/2025	Nick
VFX Implementation	DEV	12/1/2025	12/10/2025	Berke
Third Enemy + Boss Design	ART	12/4/2025	12/18/2025	Nick

Environment Art — Level 3	ART	12/10/2025	12/22/2025	Nick
Save System / Checkpoints	DEV	12/8/2025	12/15/2025	Berke
Level 3 + Boss Build	DEV	12/12/2025	1/5/2026	Berke
Rig & Animate Boss	ART	12/15/2025	1/5/2026	Nick
Interim Demo Deck + Build	ORG	11/25/2025	12/3/2025	Both
UI Visuals Style Pass	UI	12/28/2025	1/6/2026	Nick
Alpha Documentation	ORG	12/28/2025	1/7/2026	Both
Intro Animation	ART	1/5/2026	1/12/2026	Nick
Bug Fixing & Tuning	DEV	1/6/2026	1/20/2026	Both
Performance Pass	DEV	1/10/2026	1/24/2026	Berke
Final Mix / Master	AUDIO	1/20/2026	1/27/2026	Berke
Playtesting Results & Adjustments	ORG	1/10/2026	1/21/2026	Both
Build Management & Final Polish	DEV	1/25/2026	2/3/2026	Nick
Final Documentation	ORG	1/25/2026	2/4/2026	Both
Gameplay Video	ORG	1/28/2026	2/4/2026	Both

5. ASSESSMENT

Unique art and music, along with the fluency in gameplay, will be the coolest aspects of the game. And, who would play this game? Literally anyone!! Anyone who wants to enjoy their time in a caricaturized TUM environment, those who appreciate the genre and concept, or anyone who enjoys playing games in general might want to try this game.

Since this is not a serious project that will be published (at the time being), metrics other than consistently following the planned scheme and a player's flow state in the game might be somewhat meaningless for evaluating the game's success. However, if we want to list a bunch of important metrics, those would be as follows:

Core gameplay & Game Feel:

- Combat Responsiveness = › inputs are instant, no noticeable delays
- Combo Flow = › attacks are chained easily
- Hit Impact Feel = › satisfying SFX, visual feedback

Rhythm and Audio Feel:

- Reaction to music = › player notices the adaptive changes
- Sync feedback = › combos are aligned with the tracks
- Hype = › increases the adrenaline of the player

Enemy AI Metrics:

- Balance = › enemies are not too difficult, not too easy to beat
- Positioning = › enemy attacks are in sync

Players' Feedback:

- Fun = ›Did the player actually have fun?
- Replayability = › Would they play this game again?
- Flow state = › Does the player get bored? Or in a flow state while playing?

We believe that other metrics are closely tied to these main ones. To make a successful beat'em up game, players must have fun while their responses justify these metrics in the four different categories we listed.

Milestone II

1. Iteration on the Game Idea

1.1 Balancing the Fighting Scheme:

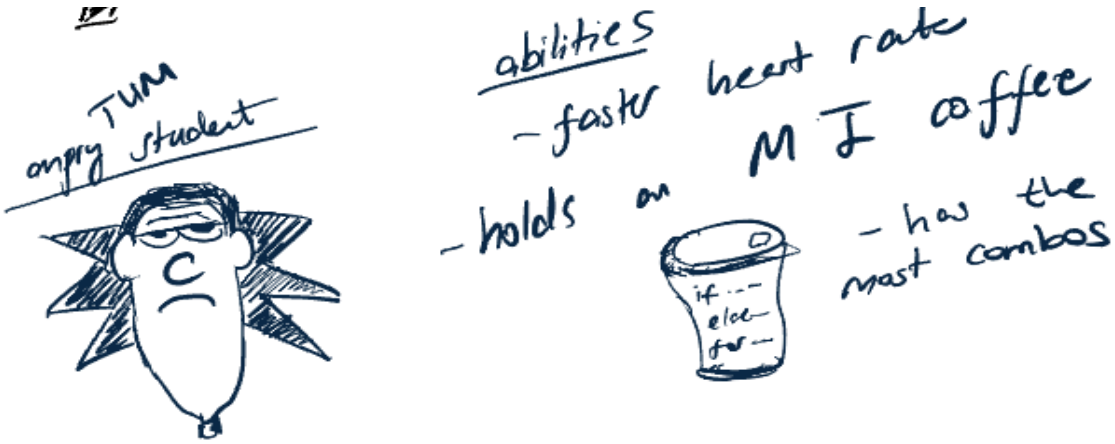
After completing the design process for the prototype, we realized that the attacking and defensive moves of the player and the enemies must be balanced. In a similar fashion to the famous traditional game Rock, Paper, Scissors, all the attacking and defensive moves must have counterplays. This, after all, is a core strategic concept used in classic fighting games, with the moves being attack, throw, and block.

1.2 New Characters:

We added three new enemy archetypes that might be revised in the future. We considered the three enemies to possess the following main abilities: **mobility**, **range**, and **tankiness**. The enemies are as follows:

1. **Angry TUM Student:**

Sketch:

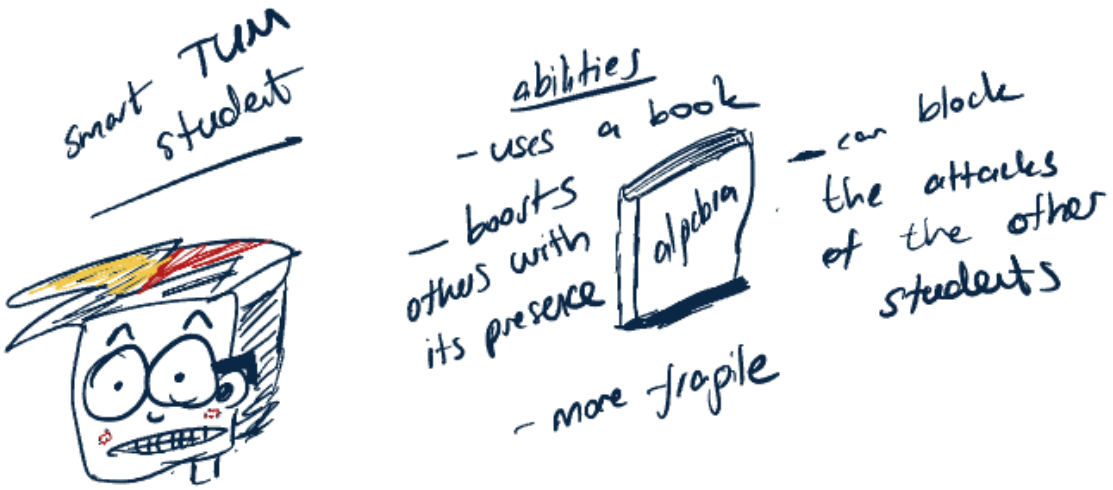


Properties and Abilities:

This character has a faster heart rate compared to the others, and their **mobility** stands out the most among the enemy types. He carries a coffee cup, which might be considered a throwable object in future iterations. He also has the biggest moveset amongst all the enemies.

2. **Smart TUM Student:**

Sketch:

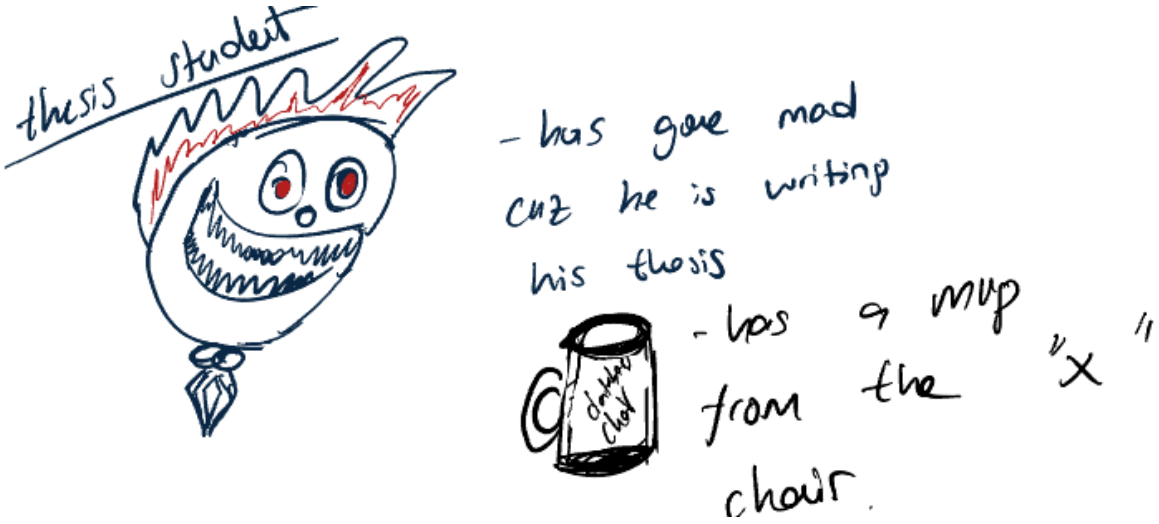


Properties and Abilities:

This character uses an algebra book (because she is very intelligent). She can block the attacks of the protagonist on behalf of other foes, even if the attacks are not directed at her. She can also boost others with her presence, giving them more attacking power. In return, this character is more fragile. Accordingly, she is classified as a **ranged** unit, comparable to a sorcerer.

3. **Thesis Student:**

Sketch:



Properties and Abilities:

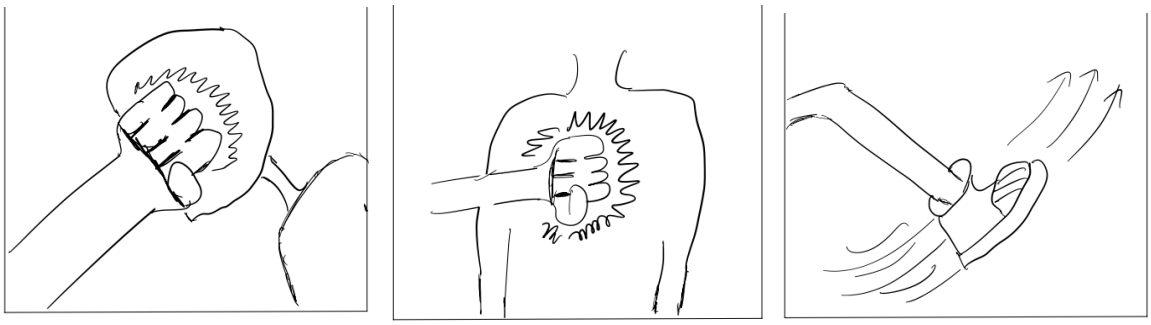
This character has gone mad because he is writing his thesis at the moment; most students would go insane writing their thesis. His attacks cause a lot of damage if not blocked, especially compared to the other foes. He moves slowly and requires a lot of hits to take down. Additionally, he carries a coffee mug from his thesis chair. He belongs to the **tanky** archetype.

2. Prototype Notebook Chapter

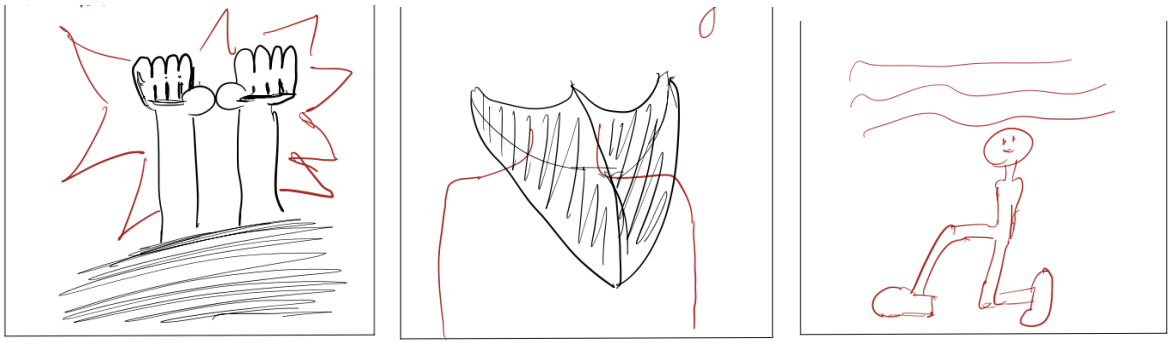
2.1 Prototype Design Idea: Lane Brawler

We envisioned a game in which a brawl takes place in the middle of MI. In our setup, we have a protagonist (a.k.a the player) with four health points, and the three enemies with only one health point each. To keep the prototyping simplistic, each character has the same moveset, which includes either **three attacking** or **three defensive** actions. These moves have sketches as follows:

Attacking Moves:



Defensive Moves:

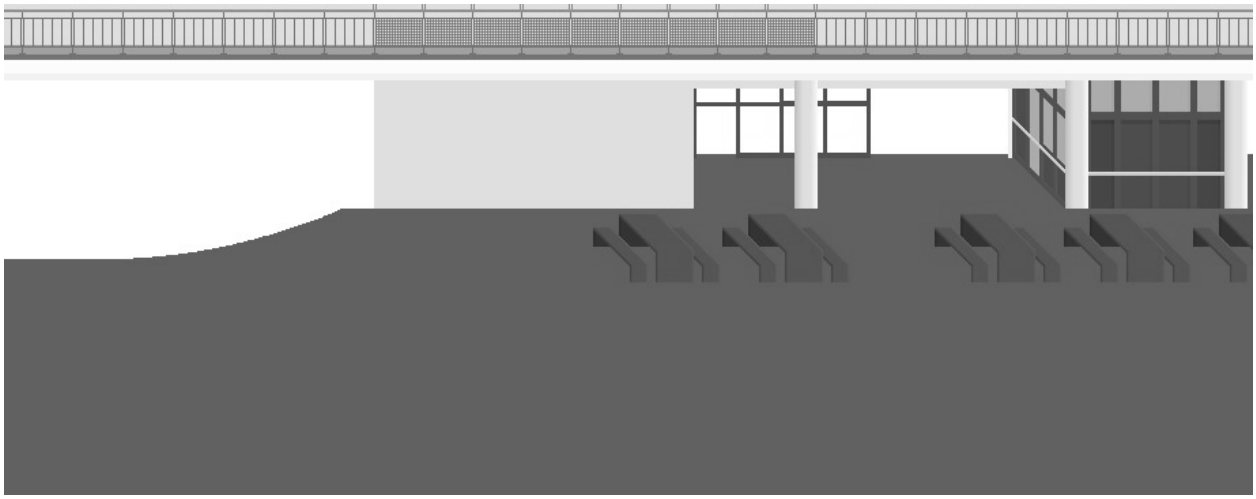


The game begins with the player acting as the attacker. In each turn, both the player and the enemy they are currently in combat with have to choose a move to make. First, the player selects one of the attacking actions listed above. The enemy must make a choice about which defensive move to make by predicting the type of attack the player will attempt. According to the moves, the encounter outcomes are as follows:

Action	Beats	Draws	Loses to
Jab	Mid Guard	High Guard	Crouch
Body Shot	Crouch	Mid Guard	High Guard
Kick	High Guard	Crouch	Mid Guard

If the player manages to beat the enemy, the enemy gets defeated in just one hit as they possess a single health point. If the enemy blocks the player's attack, the game proceeds as if the attack had not occurred. However, as an example, if the player plays the "Jab" action and the enemy responds with the "Crouch" action, the enemy gets a counterattack chance, and the roles switch. Now, the enemy attacks, and the player must defend against the enemy's attacking moves in combat. The game continues until either the player loses all their health points or all the enemies are defeated.

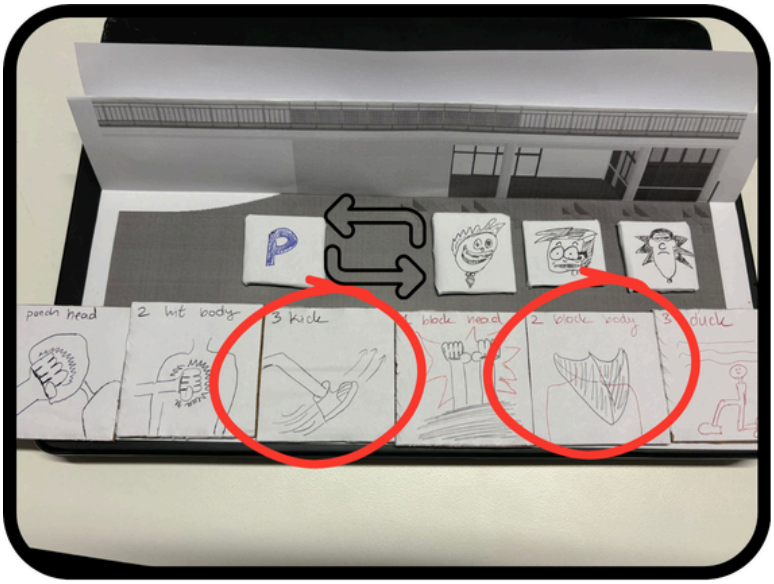
The GridWorld Sketch:



An Example Scenario of the Gameplay:

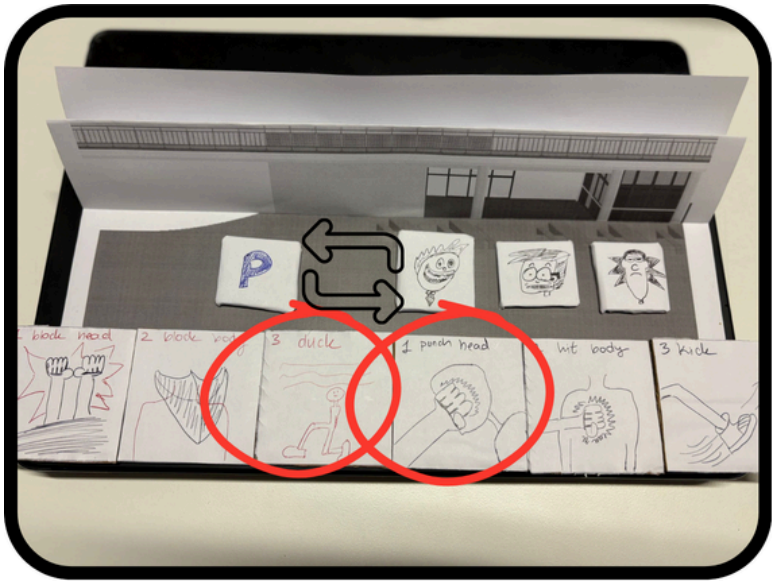
Round 1:

Player: 3-Kick
Enemy: 2-Mid Guard
Outcome: Swap the actions



Round 2:

***Player*: 3-Crouch**
***Enemy*: 1-Jab**
Outcome: Swap the actions

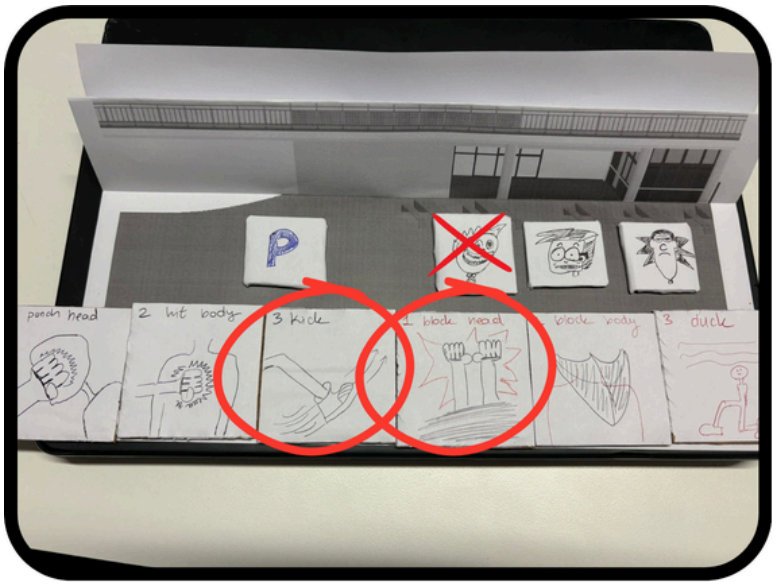


Round 3:

***Player*: 3-Kick**

***Enemy*: 1-High Guard**

Outcome: 1st Enemy is defeated

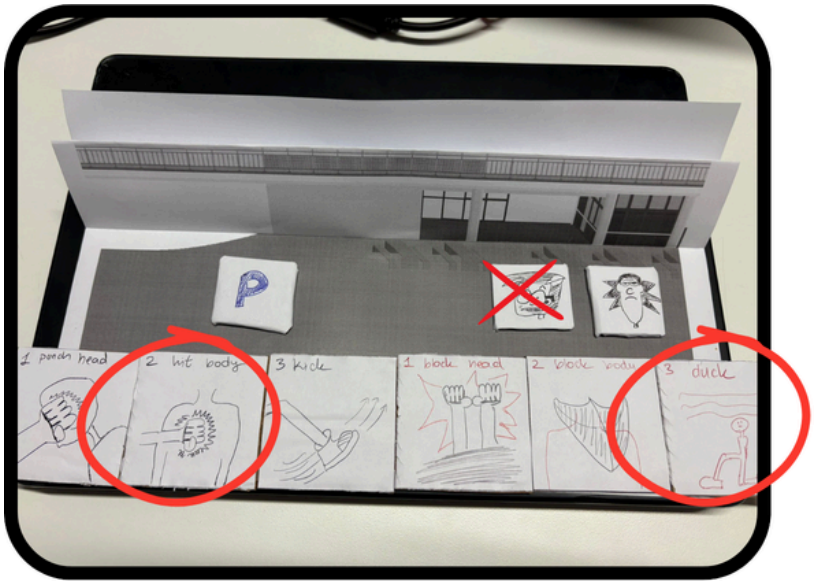


Round 4:

***Player*: 2-Body Shot**

***Enemy*: 3-Crouch**

Outcome: 2nd Enemy is defeated

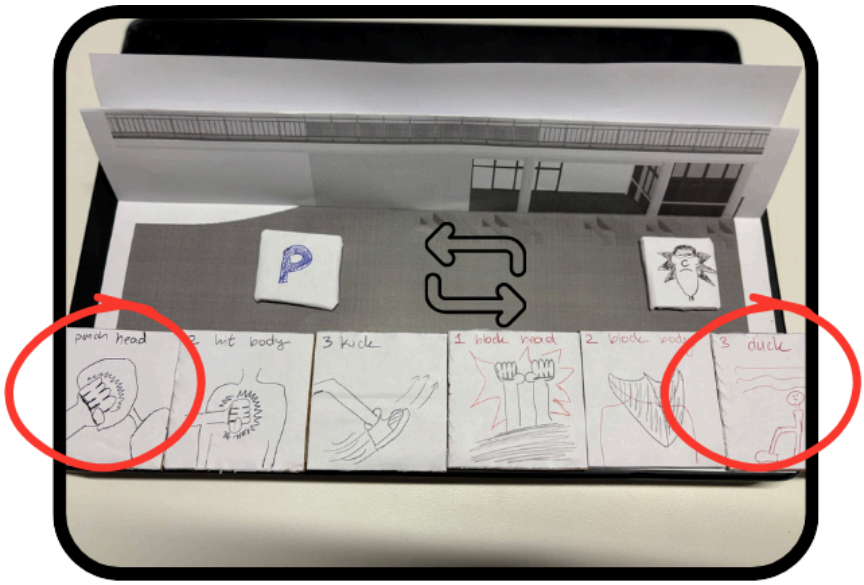


Round 5:

***Player*: 1-Jab**

***Enemy*: 3-Crouch**

Outcome: Swap the actions

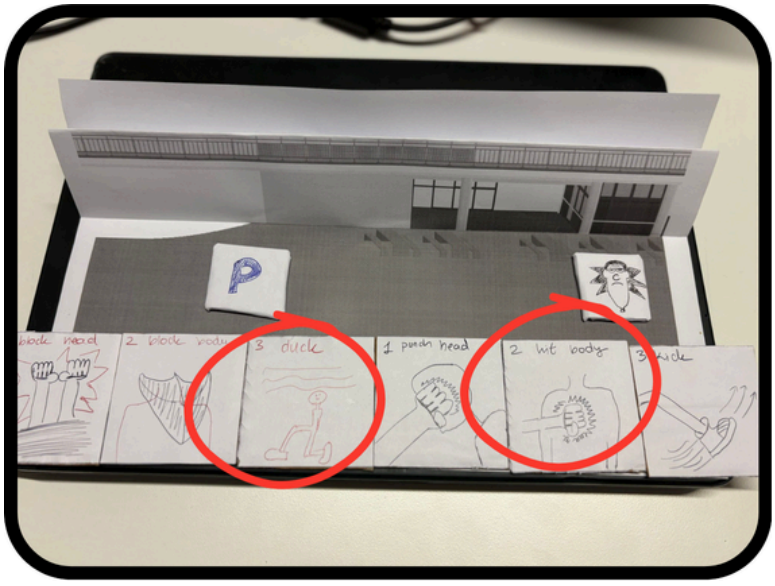


Round 6:

***Player*: 3-Crouch**

***Enemy*: 2-Body Shot**

Outcome: Player takes a hit

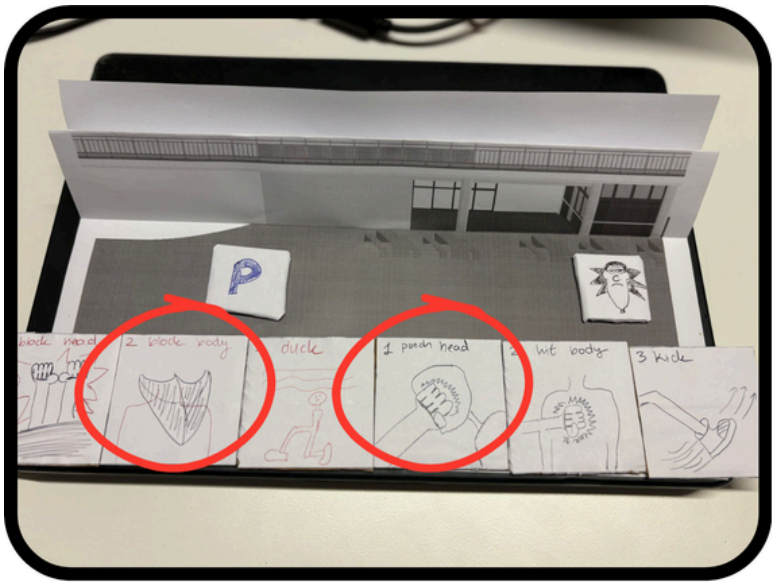


Round 7:

***Player*: 2-Mid Guard**

***Enemy*: 1-Jab**

Outcome: Player takes a hit

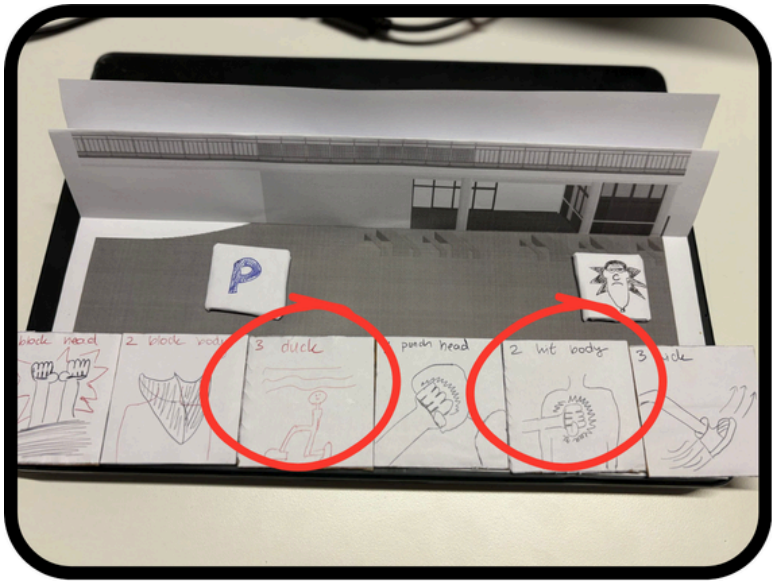


Round 8:

***Player*: 3-Crouch**

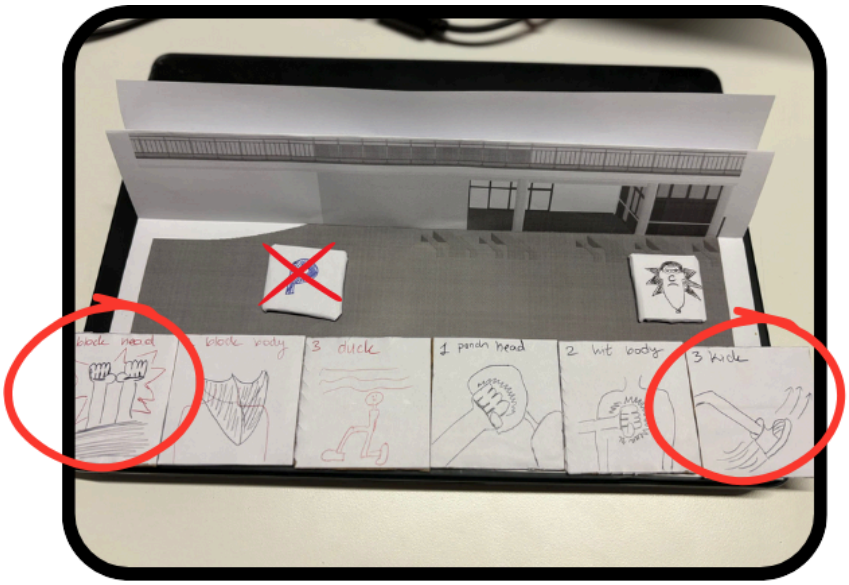
***Enemy*: 2-Body Shot**

Outcome: Player takes a hit



Round 9:

***Player*: 1-High Guard**
***Enemy*: 3-Kick**
Outcome: Player is defeated



Report on your experience playing the game:
The game, while charming due to its cartoony drawings and basic gameplay formula, felt simplistic and made us realize just how good of an experience a computer game can deliver, in which we can bring all our crazy imaginations to life. With that said, playing using our original drawings and TUM-themed assets while at the same time trying to imagine the moves we perform as an animation definitely adds to the experience and immersion. Additionally, playing the game made us realize that we need a better naming scheme for the moves, as most of these can be described by terms commonly used in fighting arts or video games. The names have been updated, but the pictures showcase the older names.

What we have learned from creating the prototype:
The simplicity of the physical prototype forced us to think of basic concepts that work gameplay-wise which can lay the foundation of the combat system in our implementation of the actual game, that of course being the rock, paper, scissors mechanic that most fighting games are based on. We have learned that we should put more time into designing and finalizing the game scheme, character moves, and combos, even before initiating the development process. Thinking about how to balance the actions, to make everything coherent in a way that was definitely harder than we thought. It is essential for us to determine the basic combat-related gameplay before reaching a point at which changes become too time-consuming. So far, this prototyping phase has not let us fully finalize the core aspects of the gameplay (at least for the first level), but we have come to the conclusion that new types of abilities for the player and the enemies must be added to the game for diversity purposes.

Milestone III

Basic Functionality

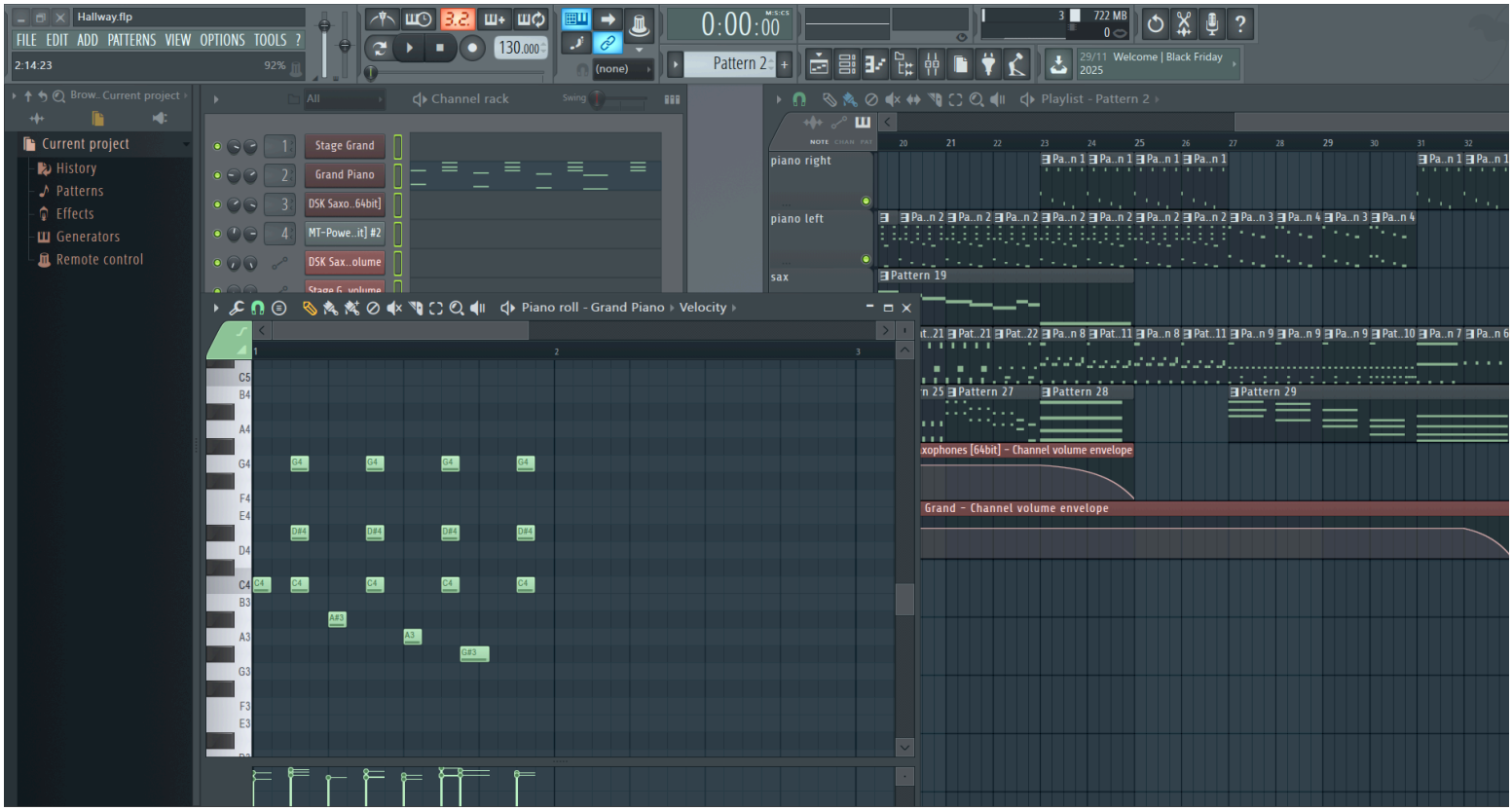
During this phase, we worked on implementing the most essential parts of our game so we have something playable in our hands. This involves the implementation of basic gameplay like moving the character and interacting with enemies, as well as creating assets such as sprites, animations, music, and sound effects, whether they are final or serve as placeholders. At this stage, we also finalized fundamental aspects of our game, such as deciding whether to use a 2D or 3D scene, setting the resolution and aspect ratio, and addressing smaller details like how to organize the asset filesystem and establish naming conventions for those files.

Importance of Assets

Having original assets that match the vibe of our game is super important. We're going for a cartoony feel, something light-hearted and fun, but still polished enough that players can get immersed and want to spend time in the world. Since the setting is a university, achieving that shouldn't be too hard, as most of the people playing the game are already accustomed to the locations.

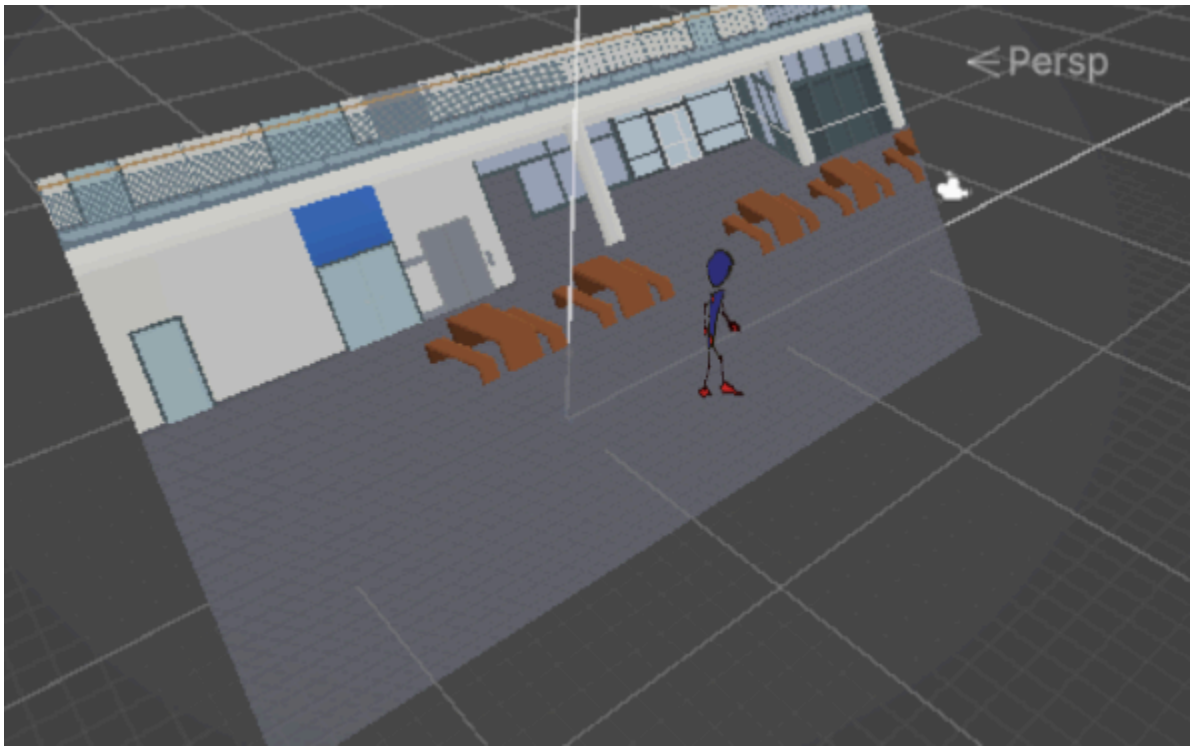
That's why we're leaning toward a simple art style, with backgrounds that loosely resemble the real locations and characters painted with thick, expressive brushstrokes in Photoshop and Procreate. It should be mentioned that the current art isn't final. Especially when it comes to proper player and enemy assets, it's a task that takes a lot of time, care, and attention to detail, so for now we're using placeholders while we work on getting everything just right.

Sound is obviously another huge part of the game. We want the music to be catchy and energetic, something you'd actually enjoy listening to while getting into a brawl. High BPM, punchy drums, and a rhythm that keeps you fired up. For our first track, we went with a jazzy vibe, almost like an 80s-style saloon fight theme. We also made a second version of the track that sounds less impactful, and that one plays when the player's performance drops. It creates this fun contrast where good gameplay literally sounds good and incentivizes the player to perform well, which is surprisingly satisfying. On top of that, we added basic sound effects for running and punching to make actions feel more responsive. All the music and sound effects were created in FL Studio, using a real MIDI keyboard to record the parts.



Choosing the Right Dimension Style

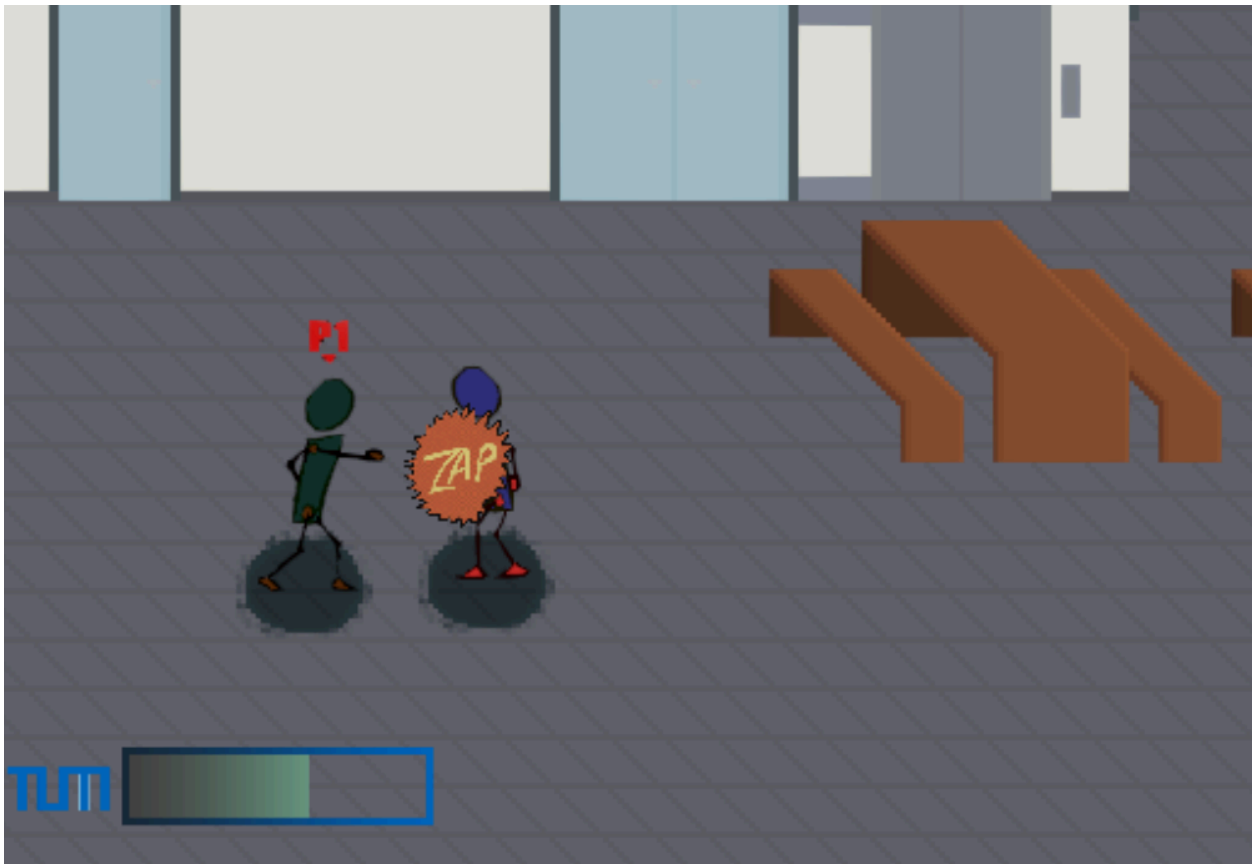
Considering we're making what's basically a 2.5D game, one where you can move horizontally, vertically, and still jump, we originally started building everything in a 3D Unity scene. Using 3D at first made it easy to handle movement on all three axes. We used an orthographic camera pointed down at the stage and placed our 2D sprites in front of it to make them look like they were standing on the ground.



However, we soon came to realize that the third axis started causing problems. Moving "up" on the stage meant moving along the z-axis, which sometimes made the player clip into the environment if they walked too much. On top of that, we're targeting a low resolution of 640×360 in the most common aspect ratio that's adapted worldwide, 16:9, and achieving true pixel-perfect rendering in a 3D setup turned out to be way more difficult than expected.

Because of all this, we eventually moved the game over to a full 2D setup. Almost all of the game functionality transferred over without much trouble, and now we can guarantee proper low-resolution projection, which is super important since our background art relies on individual pixels scaling cleanly to higher resolutions without stretching. Additionally, removing the third axis means less code complexity overall and simpler physics. The only real challenge with switching to 2D is handling the jump mechanic, which will require a bit of a hacky workaround, but it's definitely doable.

Screenshots from Gameplay



Current Technical Progress

The game in its current state is a good demo of what we are trying to achieve, but there is still a lot more to do. Considering what we promised and planned in the development schedule section of the first milestone, we are progressing more slowly than expected. However, we can confidently say that we have completed all the tasks listed in the functional minimum part. Regarding the low target goals, we successfully completed some of the promised features, such as a composed track with clear and adaptive layers. However, the adaptive layered musical instruments have not yet been integrated into the gameplay. Moreover, we have a simple UI that displays the player's health bar. We also have a basic combo counter but it’s not in a state where it makes sense to display it on the game UI as it’s still work in progress. As mentioned in the 'Importance of Assets' section of the document, we have a simple sketched character with basic animations, including idle, block, run, and punch. These animations are created with layers in Procreate and can be expanded to include the promised assets in the future.

For the Enemy, we have the Enemy Coordinator, which handles enemy behavior in the scene, such as chasing the player depending on the numberOfAttackers variable. This limits the interaction between the enemies and the player to prevent overwhelming the player with a high number of enemies. Additionally, each enemy has a script called EnemyBrain, which contains the appropriate functions for enemy actions, including state transitions from idle to approach, attack, and fall-back states. The coordinator feeds each EnemyBrain with the necessary positional information, and the EnemyBrain script ensures that the enemy animation states, hitboxes, and movement logic respond correctly to the player’s actions.

Each character in the game has a Health script that can be adjusted with an OnHealthChanged event function by whoever subscribed to that event. A character takes damage when the necessary conditions are satisfied, such as when the opponent's hitbox overlaps during a valid attack window. This causes the player's health points to decrease and triggers listeners, such as the health UI or hit VFX, to update immediately. This event-driven structure ensures that all subsystems remain decoupled which makes the logic easier to maintain and extend.

So far, we have three different manager scripts, each with a single Singleton instance. This class is accessible by any of the scripts, as it is public and is responsible for important tasks. The first manager we implemented was the HitEffectManager, which is responsible for pooling the available VFX Object prefabs. The other classes can call the Spawn function of the HitEffectManager. Each VFX Object has a HitEffect script that is responsible for setting the GameObject active in the scene and playing the animation it has connected to the object via the Animator component or playing the ParticleSystem component attached to the prefab. This pooling system allows us to reuse effects instead of instantiating them repeatedly, which significantly improves runtime performance when many hits occur in a short time. Another manager worth mentioning is the EnemySpawner. Although this class currently has simple functionality, such as spawning the first enemy according to the first beat drop right after the chorus of the music starts, and spawning the other enemies one by one whenever an enemy gets eliminated by the player, it already establishes the foundation for future level pacing, difficulty, and dynamic wave generation.

Implementation Challenges

The most challenging aspects that consumed the majority of our initial time were already mentioned in earlier sections, specifically the discussions and the decision-making process of choosing whether to develop the game in 2D or 3D. This decision had a large impact on our workflow, our asset pipeline, and ultimately the general “feel” of the gameplay. On the other hand, the NPC AI proved to be much more challenging than we initially anticipated. Creating an enemy that behaves like a smart agent, reacting to the player, coordinating with other enemies, and appearing as if controlled by a real person, is quite difficult. At the moment, our enemies have no specific functionality beyond chasing the player, idling when the player exits a specific radius, and continuously attacking once inside attacking range. Their state transitions work, but the behavior itself is still extremely simple.

It is also worth mentioning that due to several unforeseen events earlier in the semester, the start of the development process was delayed, which naturally slowed us down. Even though this setback affected our progress, the current state of our systems is promising, and we expect the intelligence and expressiveness of our foes to improve progressively as we refine their movement and decision-making logic.

Another technical challenge we faced was the implementation of the player’s combo system. Deciding on the internal architecture for combos, how inputs should be buffered, how we should

track the current combo state, and how we allow the player to branch into different attack sequences required a surprising amount of design work. Our first attempts relied on simple “input windows” hard-coded into animation events, but this quickly became messy and difficult to scale. We later introduced an Input Buffer and a structured representation of combat moves, allowing each attack to define its own follow-ups and timing constraints. Although the system is still not fully integrated into the UI and lacks visual feedback for the player, the underlying logic for chaining attacks exists and can be extended into a more robust combo tree in the future. This challenge highlighted the importance of early planning in combat design, and it will remain an area of active development as the project evolves.

Milestone IV

Implementation Challenges & Technical Details

NPC AI

Currently, our NPC logic consists of a Hierarchical Finite State Machine that has the following states:

- EnemyState.Attack
- EnemyState.Idle
- EnemyState.Approach
- EnemyState.Stunned (**currently not functional**)
- EnemyState.Scripted (**currently not functional**)

These states are controlled by the EnemyCoordinator as promised in the previous milestones. The coordinator updates the corresponding target positions for each enemy and assigns these positions to the respective enemy NPC scripts. However, NPCs can also make their own decisions and have their own decision-making logic. They are not fully dependent on the Enemy Coordinator.

So far, we have a solid structure in code that utilizes some of the OOP concepts for game development, such as the Singleton pattern. There is an Entity class and an EnemyEntity class, which derive from the base Entity Class. Enemies have their functionality distributed across several scripts, which are as follows: EnemyCombat controls everything related to combat, EnemyMotor controls everything related to the movement of the NPC, and there is a brain class that encompasses everything, which is called EnemyBrain. The Enemy Brain determines which action to take for each enemy based on the Utility values derived from those actions. Each enemy action, such as approach, attack, retreat, block, and idle, has its own script, which derives from the scriptable object base class EnemyAction. Additionally, we followed a similar strategy whilst implementing the different types of attacking abilities, which currently consist of only the Melee Strike Ability and the Throw Ability. Each action and each ability has its own utility calculation function to choose the best possible action/ability. For us, coming up with this structure was the most challenging and time-consuming part, which ultimately resulted in mediocre performance compared to what we had idealized for the NPC behavior. Some of the utility calculation functions should be assessed or possibly rewritten to ensure that the enemies behave intelligently, so to speak. It is worth mentioning that each time an enemy needs to make a decision, the NPC creates an EnemyDecisionContext Object that has the following properties, provided in the screenshot.

```
public struct EnemyDecisionContext
{
    public EnemyEntity Entity;
    public EnemyConfig Config;
    public Transform PlayerTransform;
    public EnemyRole CurrentRole;
    public float DistanceToPlayer;
    public float TimeSinceLastAttack;
}
```

Each of these variables is used in the utility function to determine a utility score, which is then clipped between zero and one for simplicity. Furthermore, the enemy chooses the best possible action by comparing these values and takes an action. Recently, we realized that the enemy AI behavior has the attribute of having “ADHD”, since it makes decisions almost every frame, which makes the whole system a bit jittery and unstable. The enemy doesn’t stick to the action chosen in the previous frame, or at least, until the action finishes its lifetime. We are currently working on resolving this issue, which we hope to finalize before the next milestone concludes.

Another problem was the solo combat with the NPCs. The flaws in the AI logic become truly transparent to the player due to the problem mentioned in the preceding paragraph, as we present the enemies to the player one by one. To mask this problem, we realized that sending enemies towards the player in batches of two or three is a better workaround solution, avoiding the complications of coding and editing the existing pipeline. To conclude this section, we can confidently say that we implemented HFSM with the Utility AI functionality with a sense of peace of mind. It’s still a work in progress and an ongoing system that we aim to refine further before the last milestone or demo day.

Boss Fight Scene

In simple words, we successfully created and met our creative criterion for the boss-fighting stage. The stage consists of two phases, in which the first one, you, as the player, are trying to beat the computers spawned by the professor. The computers experience a fault error one by one, which makes it easier and more understandable for the player to determine what needs to be done to beat the stage. After the player destroys each computer, the professor comes into play, which indicates that the player has advanced to the second phase of the boss fight gameplay.

In the second phase of the fight, the player encounters the professor. First, the professor turns around and starts charging into the five charging points on the scene, marked with yellow squares. Each time the professor arrives at a charging point, it loses the charge it has to move to the other one for a predetermined seconds (such as 3 seconds). The professor only takes damage when its charging animation is playing, and the professor only deals damage while moving to another charging point. To make the boss fight consistent with the theme we have in

the games lab, we thought of, rather than waiting for the player to beat the professor, after a number of predetermined charges to the charging points, the professor takes the time back, and the second phase of the boss fight starts again. Until the player beats the professor, this time reset functionality repeats itself. In conclusion, after the player beats the professor, the game is simply over and finishes.

Combat Controls

We gave the player four core combat controls: punch, kick, slide, and block. Each move has its own distinct animation, duration, and damage output.

The punch is the most basic of them all. It's the fastest move and serves as the foundation for combos. You can chain three punches together, or go for two punches followed by a kick to trigger a *swing kick* which serves as a fast and high-damage finisher.

The kick is a heavy hitter, but it comes with a drawback. It has a longer animation and you can't start a combo with it. In fact, if you mess up your input during a punch combo and accidentally throw a raw kick, the long recovery time leaves you wide open to get punished.

The slide can only be performed when the player has momentum. It deals a little damage and slows you down slightly as you friction with the floor, but its real value is that it's a perfect way to slip under incoming projectiles, which can otherwise be tricky to avoid.

Finally, the block negates damage from standard attacks, though it won't save you from projectiles. To use it efficiently, you have to predict the enemy's attack timing to block at the right moment.

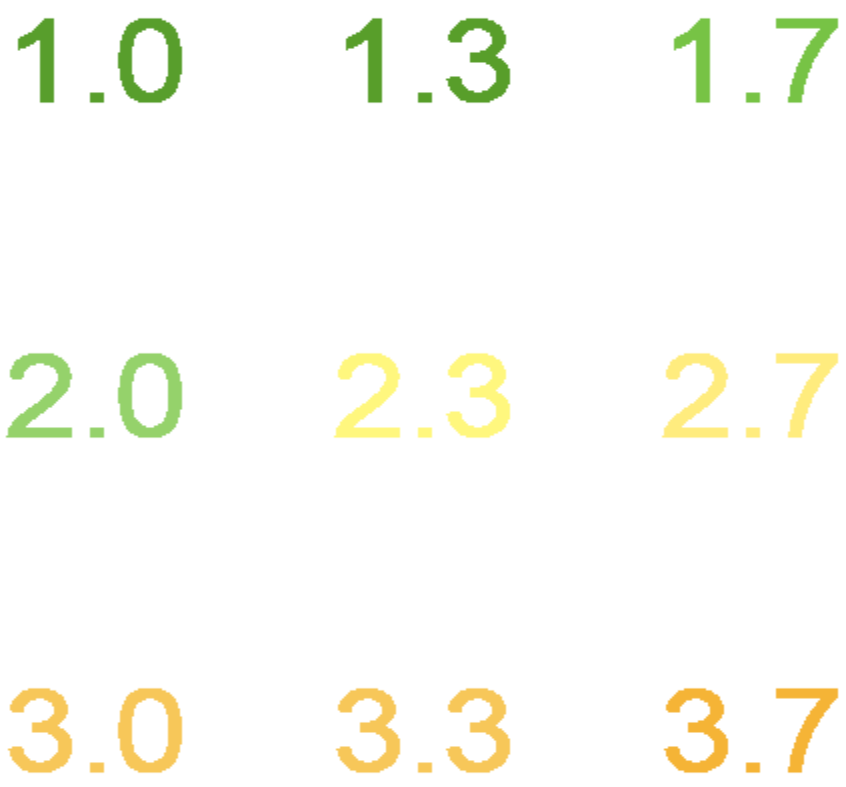
To keep combat fluid, we implemented an input buffer. You have to enter the command for the next move during a specific window close to the end of the current move's animation. We originally tried a system that let you queue up multiple moves, without any timing in particular, but it felt disconnected and over-engineered. We rewrote the logic to prioritize timing, which fits our game's pacing much better.

Scoring System

We kept the scoring system straightforward. Once you land a hit, your score goes up, and if you take a hit, then it goes down. As the score changes, it fills a number with color which represents your grade in the university. Once the number is fully filled or emptied, you will jump to the next or the previous grade respectively. It's a solid foundation that gets the job done for now, though we might expand on this idea later.

We modeled the scoring steps on the German grading system, where a lower number is better. The best possible score is a 1.0 and the worst is a 4.0. So, for example, ranking up from a 3.3 grade would make you jump to a 3.0. As for the colors of the grades, we followed the grade coloring scheme used in the TUM Campus app, in which the best grades are colored green, and they progressively get to red as the grades get worse and worse.

The music is tied directly to this performance too. If your score slips, the soundtrack starts to noticeably sound worse, with fewer instruments playing and melodies being less impactful. The only way to get the music sounding crisp again is to play better and improve your grade.



Stage and Character Assets

While coding the core functionality of the game, we realized that we were running low on time for the assets, especially the art. We expanded the main hallway to include the cafe and more of the market, but we didn't create a new area for the second level.

Additionally, this created a huge bottleneck, and we decided to adopt a faster approach, which involved giving commands to Google's Gemini to create our character assets. Other than that,

we didn’t use any additional AI-created art. This greatly accelerated the progress, and we realized that we are not artists and shouldn’t attempt to adopt the skills necessary during this project phase. It’s always better to focus on what we can do best for the sake of the project, as also stated in each milestone in this practical games lab course.

While not as catchy as our first track, we used the same piano and drum plugins to create a new track for the boss fight.

The Missing Functionalities

Although we wanted to add a second playable level between the boss fight level and the brawler level, we couldn’t find the time to accomplish this promise of ours. In our desirable target goals, we have two composed tracks, two playable levels with a boss encounter featuring a simple

phase change, HFSM & Utility AI, fully animated main character and enemy animations, refined hit responses that include camera shake, and improved environment art, all of which have been completed.

We don’t have any of the High Targets implemented currently, as the required functionality and implementation time are already a huge burden and weren’t designed to be actually achievable by the end of this project phase. Having the BPM HUD, rhythmic phase changes, and adding cooperative functionality to the game are significant tasks in their own right, and we are proud to have made this much progress in the current state of our project.