h_da hochschule darmstadt | member of eut+

# University of Applied Sciences Darmstadt

# Faculty of Media

**Bachelor of Arts Animation and Game**

**Semester 2**

**Me and My Shadow**

This game is a 2.5D puzzle platformer, where you play as a shadow figure, who can only interact with shadows on the wall casted from the foreground objects in the room, and a kid who can move said objects to manipulate the shadows on the wall for the shadow figure to progress further, in order to collect the crunchy cookies in each level.

Mike Li, 1117838

Date of Submission: 02.07.2023

## Introduction

‘Me and My Shadow’ is a puzzle platforming game with basic “Jump & Run” movement with a unique catch. You play as 2 characters in the game. The first one is a shadow figure called “Mr. Shady”, who can platform on shadows casted on the walls from room objects and even push their shadows to move them in the foreground. The second character is a kid named “Sam”. He is able to grab and manipulate room objects, which will result in shadows changing its position, shape and scale by moving, rotating, pulling towards and pushing back the objects. Each Character can be played with different input methods. Mr. Shady can only be played with the keyboard, while Sam can only be played with a mouse with its hand cursor, which also makes it possible for “2-Player-Cooperation”. My part of this game is the role of game programming in Unity, which, in this context, sounds aggravating. How should you make a system, that shadows, casted from foreground objects and a light-source, become boundaries with collisions, have minimal performance and interact differently depending on the object Mr. Shady can manipulate, without spending too much time and having enough knowledge about implementations. Therefore the explanation will be about my solution to the problem, how to tackle a seemingly complex problem and keeping in mind which visions and limitations the project overall has.

## Individual Contributions

In summary my individual contributions are as follows: 15% Animation & Game Design, 75% Game Development/Technical Art, 10% Methodology.

Animation & Game Design (15%): Involvement in design can be explained with me being the lead game designer. I worked on first basic concepts for level design mechanics, helped and collaborated with level designers and mainly did the UX design, which includes fine-tuning game mechanics/game feel and Menu UI placement.

Game Development/Technical Art (75%): This is the main part of the involvement in the project. I programmed every important mechanic for the game and level designers, to name but a few: Jump & Run movement; Grab, move, rotate, scale objects with its shadow; Interaction with shadows and objects; Automation System for Level Designers, which made it easier for designers to place objects and automatically generate shadow

objects; Sound and Particle Effect Systems; Scene Management. For Technical Art I did the Animator and handled the Git-Lab Version Control.

Methodology (10%): This percentage is the sum of two parts. For one being the Project Manager, who has to keep dates in check and discuss priorities with others with techniques like Milestone Planning Framework, Prioritization Technique (MoSCoW) and Defining MVP (Kunkel, 2023). For Second writing this document.

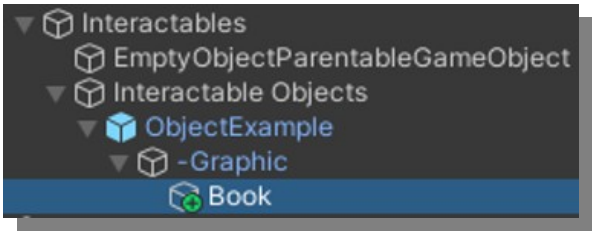
## Analysis

As said in the Introduction, the topic of the analysis chapter are my approaches and solution of how to create a shadow collision system, based on foreground objects casting their shadows on the wall. First of all, we need to consider the scope and therefore the limitations. Our Team decided, that the game will be in 2.5D, meaning that the Camera will be placed and moved like a 2D platformer game, but there are two planes to play with: The background with focus on “Jump & Run” and the foreground with focus on ‘Puzzle/Point & Click’ like gameplay. Also, due to the scope, we will only use one directional light-source, meaning there will not be an angle to calculate for shadows distorted on the wall. Because of these limitations, there is already a lot of hard and time-consuming work to ignore. All we need is a system, with which you can move foreground objects and also move its shadow along with it. That being the case the implementation consists of this idea: Creating at least two game-objects with Colliders: a foreground object and a shadow game-object. One of them is the parent, equipped with a rigid-body component, and move the child due to their hierarchy, and keeping an offset between them, until the shadow game-object reaches a wall.

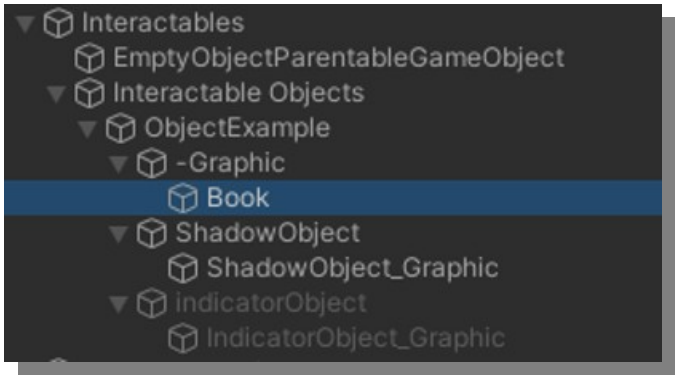
Screenshot of two objects: foreground and shadow object

The next decision is regarded to how to set up the hierarchy. By the time, I had an other problem to think through. Caused by our concept and core loop, Sir Shady needs to be able to move the foreground objects through their shadows. Respecting that, the hierarchy used to be the shadow as the parent of the foreground object. That worked for Sir Shady pushing objects away from him with rigidbody physics, but led to the next obvious question: The mouse will be able to grab, move, rotate and even pull towards and push back room items, as a consequence manipulate its shadow accordingly and with all that not using rigid-body physics, but transformations. So how should a child object move the parent. The answer was to switch up the parents. In Unity there is a neat function to set the parent of one child as an other one (Unity Technology, “Transform.SetParent”, 2021). So my solution was that anytime the hierarchy needs to change itself (in this case, every time the mouse wants to access a foreground object and manipulate it), the child becomes the parent of its parent and vice versa. Not only does it sound confusing, it also was a mess to work with so many re-parents. This approach was at first chosen, because I had no idea about the fact, that if the parent has a rigid-body component, the collision of the child will also affect the parent, unless the child also has a rigid-body component. Thus the implementation was simple: Foreground objects are the parent of the shadow object, while both of them having the same colliders but the foreground objects having a rigid-body component. Generally speaking was that the whole solution to my problem. I created a

fake system with only game-objects, which is easy to use and works good enough for the scope of our project. Knowing about this solution enables the ability to create an easy and helpful automatic tool for the level designers to create the shadows, with only having the foreground objects placed into the scene. It helped the designers immensely with creating a lot of levels quickly and boosts efficiency for their workload.

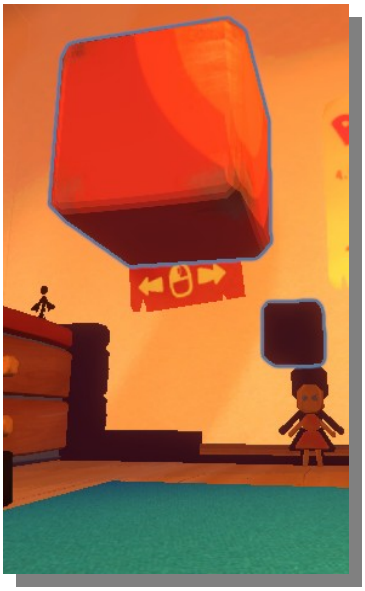


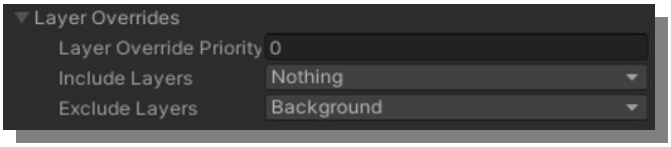
Screenshot of set-up in the hierarchy of an interactable object before scene is loaded



Screenshot of hierarchy of one interactable object with automatically created shadows

There a was problem with shadow game-objects not intersecting with the ground of the background, but being on top of them. This results in e.g. foreground objects floating in midair, because of the shadow object not passing through background's ground. This should not happen, because only the foreground objects should manipulate shadow objects, especially its location.The quick fix is by tampering with the layers settings (Unity Technology, "Collider.excludeLayers", 2021) of the shadow object's colliders. In there you can exclude layers, in other words disable calculations with other colliders with selected matching layers (Unity Technology, "LayerMask", 2021).


Screenshot of foreground object being mid-air because of the collision of the shadow



Screenshot of layer override settings of collider of a shadow object

There are also the mouse game mechanics to analyze. The Grab and Move mechanics were easy to implement. All that was needed was a ray-cast in direction from the camera

to the mouse and select the game-object with the specific layer. Moving requires a method that uses a Camera function called ‘ScreenToWorldPoint’ and moves the game-object with their transforms accordingly (Unity Technology, “Camera.ScreenToWorldPoint”, 2021). Rotation was also no problem. There is an occasion where a re-parents is needed. In fact every time, the rotation should occur, the shadow object chooses an other parent in order for the parent to finish its rotation to then apply its parent back to the foreground object. Of course the shadow object rotates itself with the same direction a the foreground object. The reason being that I do not want any location deviations to occur, while they rotate along with the pivot being the foreground object with an offset, in the other direction. Also the rotation did not need to care about with which axis the object needs to rotate with locally (Unity Technology, “Transform.Rotate”, 2021). The implementation always uses the world space transforms, meaning the automatic conversion of the world rotation axis to the local transforms of a game-object (Unity Technology, “Space.World”, 2021). This solved a lot of headaches for the rotation of a game-object and its shadow, but there was a problem with scaling. For now the scaling is implemented with a value, which adds or subtracts the current value of scale of the shadow, while the foreground object moves itself back and forward. Also for the purpose of preventing intersections between objects, I tried to also move up the shadow object from its local origin position dependent on how much the object was scaled. The approach was not a really realistic one, but it works for the most part, especially because the level designers already used them to create various levels already and time was already pretty advanced, so we scraped a more realistic approach. Another problem with creating the realistic approach is the implementation itself. Realistically speaking, if we want to track how big a shadow should be dependent on the distance between light-source, object and wall, then the best way to find out is using the ‘Intercept Theorem’. The problem with this theory is that I needed a way to at least determine the distance between the ground and the pivot of the foreground object in order to scale and also move up the object, so that it still touches the ground with a calculated offset, which is without any indicators not really possible.

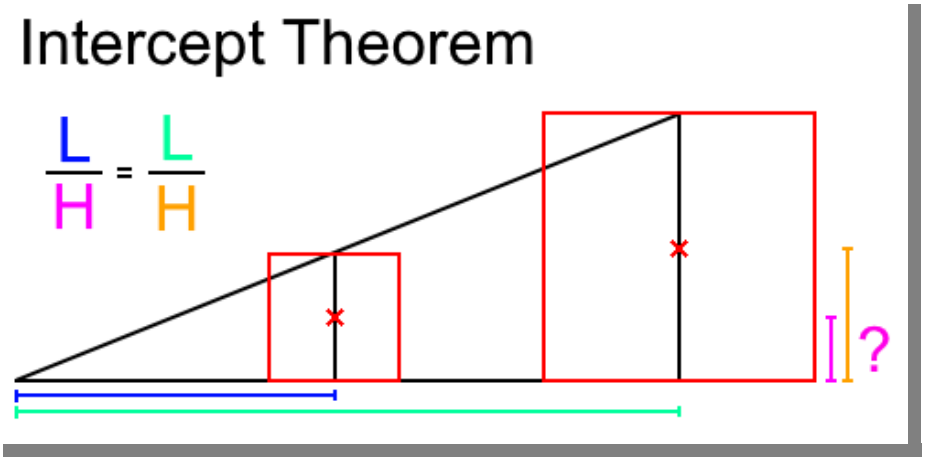
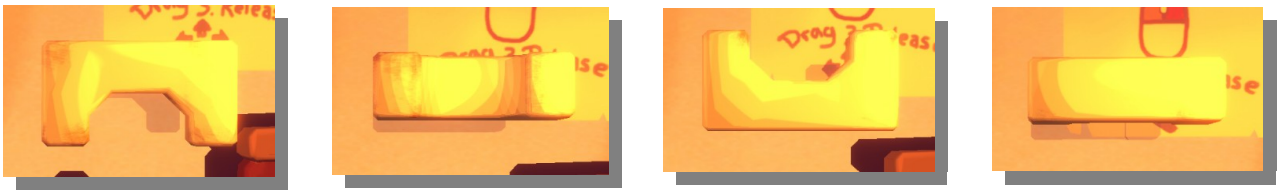


Illustration of Intercept Theorem with showing the missing part of the formula being the height to calculate the size of the shadow object on the wall

It is not practicable to either move the pivot to scale it up to neglect the offset, because we rotate the object, which makes the relocation of the pivot redundant, or to shoot a raycast downwards to calculate the space between pivot and ground, because the item can be scaled in the air leading to an incorrect value. Also the problem of rotating is, that the offset from the ground differs on how the rotation of an item is, which makes it impossible to take only one offset and use it for all rotations. With that being said, a state-machine could be probable, but a huge mess to work with due to the sheer amount of states that have to be implemented for each rotation and the direction of the axis.


Screenshots of showing that each rotation has four other rotations in an other axis. A lot for all directions.

The most feasible answer would be to create six game-objects for each item and compute the gap dynamically. As stated before, there was no time and would be a destructive workflow for the level designers with the advanced time to make it a reality. Overall the mechanics worked as intended and reached in any case the rate minimum of our projects MVP.

## Conclusion and Future Work

This is my first time working on a project with three other people, so this experience itself is very new. Nevertheless there are more specific things to mention, like the worth of making automation systems. Automation saves a lot of time due to splitting up more work for the designers to actually design the game. This project already used these systems for the primary mechanic, that being shadows being created dynamically, but there can be done more. My intention is to learn about creating more tools, which makes it more accessible to create and fine-tune in the Unity Editor itself. The last thing implemented feature of this topic used animation curves to make it more accessible to define the size of particle effects dependent on airborne time, which is very interesting and has a lot of potential for combining design and code in my opinion (Unity Technology, “AnimationCurve”, 2021). Layer-based solutions are not only very helpful in this project, but also can be important in other projects to manage, which game-object can interact with each other without using the string based tag-structure. One last major thing was the interaction with other team mates, while working on the same project. While working with GitLab was pretty much solid and working, despite getting a headache most of the times by merging different versions together, which has to be done with much caution in order to not delete any progress. Especially when at least two people work on the same scene, which is the most troublesome situation with managing with a version control. What I have heard, was the ‘Multi-Scene-Editing’ (also known as ‘Multi-Scene-Workflow’) approach (Pêcheux, 2021), where each category of game-objects will be split up using additive scenes (Unity Technology, “Managing Projects with Multi-Scene Editing.”, 2021). The advantage of this technique is the reduction of merging conflicts by splitting up scenes with the specialty of each team members duties. This could prove very helpful for all organized and structured future projects, especially for working with multiple people.

**List of References**


- Kunkel, Boris. 2023. A+G Methodology: A+G Producing + Production Management 2 (Summer Semester 2023). Dieburg
- Pêcheux, Mina. 2021. “Using a multi-scene workflow in Unity/C#.” Medium, September 28, 2021. https://medium.com/c-sharp-progarmming/using-a-multi-scene-workflow-in-unity-c-2ae9a6dc6096

Unity Documentation:

- Unity Technologies, 2022. “Transform.SetParent” Unity Documentation. https://docs.unity3d.com/ScriptReference/Transform.SetParent.html
- Unity Technologies, 2022. “LayerMask” Unity Documentation. https://docs.unity3d.com/ScriptReference/LayerMask.html
- Unity Technologies, 2022. “Collider.excludeLayers” Unity Documentation. https://docs.unity3d.com/ScriptReference/Collider-excludeLayers.html
- Unity Technologies, 2022. “Camera.ScreenToWorldPoint” Unity Documentation. https://docs.unity3d.com/ScriptReference/Transform.SetParent.html
- Unity Technologies, 2022. “Transform.Rotate” Unity Documentation. https://docs.unity3d.com/ScriptReference/Transform.Rotate.html
- Unity Technologies, 2022. “Space.World” Unity Documentation. https://docs.unity3d.com/ScriptReference/Space.World.html
- Unity Technologies, 2022. “AnimationCurve” Unity Documentation. https://docs.unity3d.com/ScriptReference/AnimationCurve.html

Unity Learn:

- Unity Technologies, 2021. “Managing Projects with Multi-Scene Editing.” Unity Learn. January 30, 2021. https://learn.unity.com/tutorial/managing-projects-with-multi-scene-editing#

## Appendices

Important source codes in the game folder location: ‘MeAndMyShadowProject\Assets\ Scripts’ will be listed here:

- MouseControls.cs
- AutoLevelBuilder.cs
- FigureCharacterController.cs
- PlayerParticleSystem.cs

No third party assets are relevant for this document.