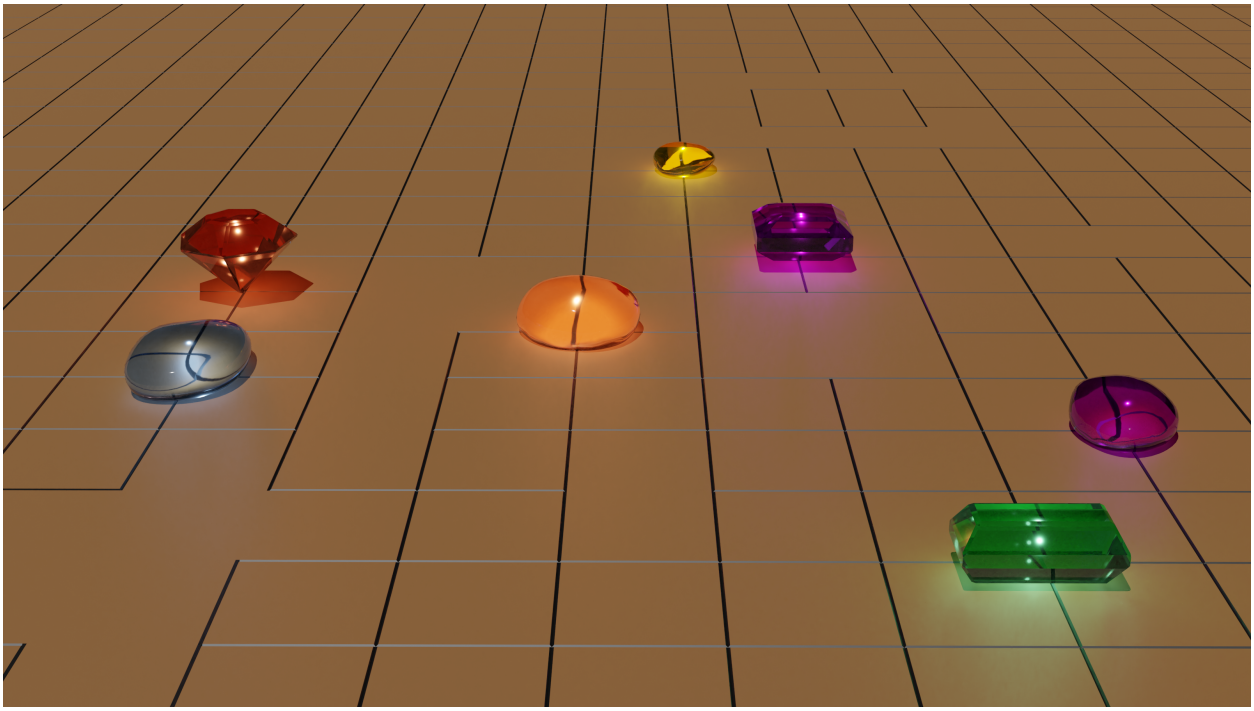
Game Pitch: *Gemji*

Team *DreiKopf*:

Felix Brendel
Jonas Helms
Van Minh Pham

April 2021



Contents

1 Game Description

Gemji is a single player puzzle game. The goal of each level is to maneuver a set of Gems across a two dimensional board to specific finish tiles. The level is considered complete when all finish tiles are occupied by the correct Gems at the same time. We initially focus on developing the game for PC. Depending on how on how tight our schedule is we may implement a mobile port.

1.1 Game Design

The game features different types of Gems with different effects that activate after a Gem is moved. These effects can for example cause specific movements of adjacent Gems. The strength of these effects varies, e.g the number of tiles a Gem is moved. As effects trigger on movement, a chain reaction of different effects can be triggered. Depending on the level we also intend to assign effects to specific tiles on the board in a similar fashion.

We created a quick example animation of the gameplay we have in mind, which you can watch in a video we have added to our wiki (gemji.mp4).

1.2 Example effects for the Gems

- Black: Purely an obstacle. Cannot be moved by the player or other Gems.
- White: Cannot be moved by the player, only due to the effects of other Gems.
- Red: Pushes away other Gems that are next to it after being activated.
- Blue: Will always move back to its original position after being moved by another Gem effect. When it is moved by the player it will stay in the position.
- Green: Pulls Gems that are positioned next to it towards itself after being activated.
- Orange: Pushes itself from other Gems or the level boundary if it was activated.

1.3 Visual Clarity

As players should be able to think ahead more easily, we want to ensure that effects attached to the Gems can be identified instantly. For that reason we plan to add specific visual effects that represent a certain class of Gem ability. Furthermore we are thinking about adding tooltips to the Gems, that show up when moving the mouse over them, which are not invasive to the gameplay but still help the player to learn the Gem mechanics.

1.4 Campaign & Level Design

The levels themselves will be designed by hand to ensure a suitable difficulty curve for the players as they learn the game concepts. The levels will be structured in a campaign that will introduce new Gem types and the dynamic map effects in a step by step fashion. We further plan as one of our high-goals to have levels procedurally generated thus creating an endless play mode that can be played after finishing the campaign.

Figure 1: This is roughly how we envision *Gemji*. See our wiki for the full animation (gemji.mp4).

1.5 Order & Chaos

The way we want to incorporate the theme of *Order & Chaos* is in relation to the gameplay mechanics. One movement of a Gem can easily cause a chain reaction that may seem chaotic initially but the effects are deterministic and the Gems clearly indicate which exact effect is assigned to them. A light-up indicator also highlights the order of execution of the effects. Therefore *Gemji* is a game which has the goal to find the correct order, so occupying the finish tiles with the correct Gems but may be perceived as chaotic in the execution of the

2 Technical Achievement

2.1 Introduction

The central secondary big bullseye idea for our project is to develop our game idea in our own engine. Our group always wanted to build their own game engine from scratch and we thought that this practical provided the perfect opportunity to put this into reality. We already started working on this engine in the winter term practical and developed a puzzle game from scratch. We want to use this practical course to further expand on the engine and see how well we can adapt it to our new game.

2.2 Motivation

The main motivation to build our own engine stems from the fact that we believe that we can reduce the overhead and therefore provide better optimization for our games on all levels of the engine, from the graphics pipeline to resource allocation and automatic memory management. Furthermore we believe

that building a game engine from the ground up presents a perfect learning opportunity, especially when trying to find suitable optimizations that fit our design philosophy.

2.3 Game Engine

In the following sections we will provide a small overview of the components of the game engine that we want to develop for this semesters project and how we try to optimize these. Furthermore we will go over the features of the game engine that we will most likely tackle in the follow-up project and how we solve the interim solutions for this semester's game.

2.3.1 Graphics pipeline

The game engine will use the Vulkan Graphics API to implement a rendering pipeline. Vulkan is a relatively new API developed by the Khronos Group (maintainer of OpenGL) with a focus on overhead reduction and was released in 2016. Vulkan provides a low-level control over the rendering process when compared to other Graphics APIs and has several advantages that also align with our overall philosophy in the design of the engine:

- The ability to run on all operating systems and devices
- Explicit control over memory management
- Decreased CPU workload due to reduced driver overhead and batching
- Making use of the driver independent Vulkan Loader to access Vulkan API entry points

The Vulkan Loader is responsible for transmitting Vulkan API calls to the appropriate graphics driver. This means that we just have to connect to the Vulkan loader in our engine and do not have to worry about drivers. Furthermore we can pre-compile our shaders into the SPIR-V binary format instead of compiling the shaders at runtime. This allows the use of a larger number of different shaders per scene and reduces application load times.

2.3.2 Overhead reduction in the engine

The game engine is developed in the C++ language that all of our team members are familiar with due to our TUM Bachelor courses such as Game Engine Design. We have also taken further steps into the direction of our core concept of overhead reduction by omitting parts of the C++ standard library, that perform memory allocations.

2.3.3 Resource Loading & Automatic Memory Management

To increase the performance of the engine we want to make sure that the loading of resources such as a texture map or a mesh is never done redundantly, which is likely the case in a puzzle game as key components are similar between different scenes. In order to implement this we allocate buffers upfront to store all our resources and a hashmap that maps the file paths of the loaded resources to their pointers in memory. If a resource becomes necessary in a scene, we can cross check whether the file path has already been loaded and then reuse the already loaded file instead of reloading it. This means that we will only load the difference between two levels which will reduce load times and create a smoother gameplay experience for the player. The hashmaps also provide further advantage for the memory management as we can free the memory and GPU memory for the texture resources by iterating over the hashmap and can incorporate this in the scene load/unloading process.

2.3.4 Sound System

Sound is very important to our design goal of creating a puzzle game as we believe that it has a relaxing or even focusing effect on the player. We will use the already existing sound system from last semester that is developed with the help of the IrrKlang library. We want to further expand on our already implemented functions to play sound effects and looping background music to also enable smooth fading between tracks and triggering a natural change for the music in response to specific game events.

2.3.5 Physics System

The current point of view in our team is that we will not implement a physics engine as part of this semesters project as it would exceed the scope of the engine building aspect. We will instead use keyframe animations and bake the limited number of physics interactions directly into the animations or generate them procedurally. This also comes with the advantage of having a tighter control over the Gems. The interactions of the Gems should not be simulated in a physics based fashion but rather deterministically executed.

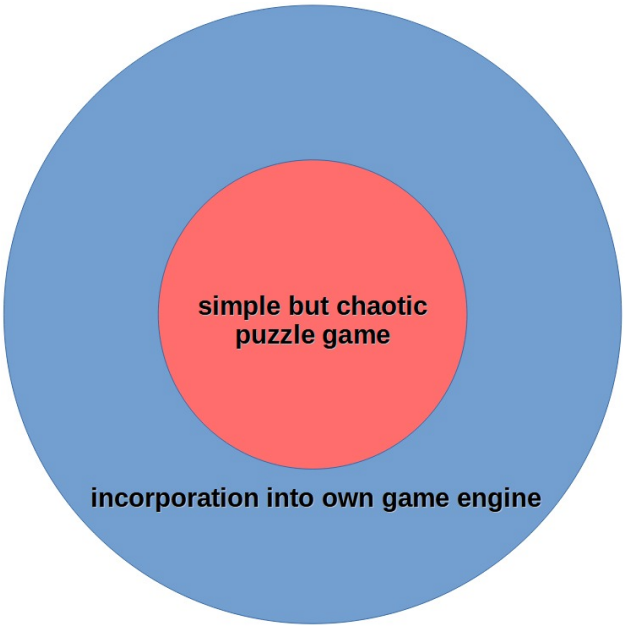
2.3.6 Animation system

The animation system will be a very important part of the engine as it will substitute our physics interactions and help to increase the graphical fidelity of the game. Implementation of the animation system will start very early on and the core functionality of keyframe animation will be finished for the interim demo. We aim to implement a system where actions and animations can be scheduled in advance to be able to deterministically describe the Gems movements.

2.3.7 Particle System

We feel like a particle system and thus interesting effects can add to the correct vibe of the game, especially in puzzle games in which you can easily get lost in the mathematics and logic of the problems instead of the game world. At the same time we have to be careful that these effects do not feel overbearing and contribute to the overall feel of the game. Sadly we were not able to implement a particle system for last semester's project which is the reason why we want to increase our focus on this aspect for this practical course and have already alotted time for it in the schedule.

3 Big Idea Bullseye



4 Development Schedule

4.1 Layers of Development

1. Functional Minimum:
    - One simple level
    - Basic selection and movement of a Gem
    - Finish condition for a level
    - One sample Gem effect
2. Low Target:
    - Effect(s) for Gem types
    - One level designed for each type
    - Sound effects & Music
    - One sample map effect
3. Desirable Target:
    - Visual clarity for effects & tooltips
    - Visually appealing particles
    - Dynamic board effects
    - Full campaign progressively introducing game concepts
4. High Target:
    - Procedurally generated levels

5. Extras:
    - Mobile platform

4.2 Tasks

1. Functional Minimum: a. First simple level
    - No "obstacles"
    - Easy finish tiles/conditions
    - Only a few Gem types

    b. Basic selection and movement of a Gem
    - Multiple Gems instead of one (qubi)
    - Targeting positions on the board
    - Targeting other Gems
    - Movement as a do_{slide} for 1 field?

    c. Finish condition for a level
    - Structure of game state considering
        - Multiple Gems
        - Dynamic Gem & map effects

    d. One sample Gem & map effect
    - Set up game logic for effects
    - Already implement the logic considering that effects may change during a level
2. Low Target:
    - Effects for different Gem types
        - Design and test effects with physical prototype
        - Implement the effects
    - One level designed for each type
        - Design levels
    - Sound effects & Music
        - Write first track for the game
        - Implement seamless change of tracks with IrrKlang
        - Look for fitting SFX or create own
3. Desirable Target: a. Visual clarity for effects & tooltips
    - Specific shaders to highlight turn order of Gems
    - ImGUI pop-up tooltips at Gem locations

    b. Visually appealing particles

- Particle spawner

c. Dynamic board effects

- Design board effects that would enhance the current set of effects

d. Full campaign progressively introducing game concepts

4. High Target:

- Procedurally generated levels

5. Extras:

- Mobile platform

The planned timeline can be observed in Timeline.png.

5 Assessment

The goal we set for ourselves with this project is creating a game that is easy to get started with, which can be played whenever the players want to relax in a casual setting. A game that is intriguing by its simplicity but at the same time complex on a level that requires tactical thinking and decision making to succeed. This is our interpretation of "order and chaos" – restoring order in a system that is defined my simple rules that spiral out into chaos. It's a game that gives the players the space and time they need. It is not about solving a level as quickly as possible but rather about letting the players feel accomplished when they solve a level. The focus is more on creating an environment in which the players feel relaxed and cozy when they play our game.

With these goals we are confronted with a few design issues that we need to solve. With a puzzle game that is intended to be learned by the players without invasive tutorials, players can easily feel lost, or get the feeling, that they do not yet have all the necessary knowledge to solve a puzzle and get frustrated. We will have to find a way to keep any frustration to a minimum. With the game design we intend, it would be possible for a level to become unsolvable after a wrong move. We will therefore introduce a undo and restart feature that lets players restart the level or undo the last move they made. Making mistakes is part of solving a puzzle and we do not want to punish the players. They should be able to try out their ideas and if they want to go back, they can.