Project: Match-3

Design Document

Introduction

The Challenge

I was challenged by Miniclip to develop a Match 3 game in C++ (or Objective-C) using the libSDL library. The rules of this tech task state that I can use additional libSDL libraries – such as SDL_image to load images, SDL_mixer to play sounds, and SDL_ttf to use TrueType fonts – but no pre-built game engine. This document serves as a design guide for myself and as a summarized piece of documentation for the game.

I have never actually worked with SDL itself – although I have developed a game engine using OpenGL in the past – so this is a great time to finally develop a game with the library. Furthermore, I wanted to take this project as an opportunity to learn more about data-driven programming and implement an ECS architecture (more on that later), which I have been wanting to delve into for a while now.

There are no small projects. Let 's get started!

Gameplay requirements

The main objective of this task is to implement the basic Match 3 gameplay according to the following rules:

- The game area is an 8x8 grid containing objects of 5 possible colors.
- The player can swap two adjacent tiles – horizontally or vertically – by clicking on both objects or swiping from one to the other.
    - If a swap results in a sequence of three or more objects with the same color, those objects disappear.
    - Otherwise, the objects are switched back to their original positions.
- Anytime objects disappear, the objects above newly empty spaces fall down, and new objects spawn from the top of the grid and fall, filling the empty spaces on the grid.

Additional goals

We have three critical factors to evaluate the quality of our product. First, its **readability** and adherence to **good development practices**. I like to think of it like this: *If I were to stop working on this project, and someone else were to pick it up and continue development, how easy would it be for them to do so?*. It is one of the reasons for writing this very same document is to provide myself and others with a simple, focused way to understand my design choices and objectives.

Secondly, the **maintainability** and **extensibility** of the product. This will be the focus of most of my architecture decisions. It's good to produce something that works for an intended purpose, but it's great to produce something that *will* work for future intentions. We don't want to simply implement a bare-bones match 3 game – we want to be able to improve upon it with additional features. Here I'm already thinking: *What if I wanted to create a level with 2 - or several - grids on screen?; What if I were to up the number of grid objects from 64 to 640? Or 6400?*. I already know we will want to add special objects with specific abilities, we might also want objects with no color or objects that can't be moved by the player. We cannot not consider all possibilities, but we can try.

Lastly, and most importantly, we must focus on the **user experience**. That is what it is all about in the end – to provide a great experience. For this, we focus on gameplay rules, on the program's performance, on how information is provided to the player, on aesthetics and feeling. Here I am going to focus primarily on a smooth-running game with a great-feeling swapping mechanic. The clearing and dropping of tiles must feel satisfying and be fast to keep the action going. On a later stage I will worry about defining an aesthetic and implementing features such as more complex animations and sound effects.

Research

It is always good to do a little research. Let's investigate the competition on the market and see what we can learn from them.

Bejeweled Classic

The king of arcade. Bejeweled sticks dearly to the classic formula of matching tiles in a 8 by 8 grid, focusing on the core user experience of destroying gems, and introduces arcadey gameplay elements to keep the player interested:

- *Classic* and *Zen* modes are casual modes which incentivize the player to turn down their brain and simple match tiles at their leisure.
- *Lightning* and *Diamond Mines* (and more recently *Ice Storm*) modes introduce a time limit and ways to extend said time, with a focus on playing as fast as possible in order to beat high-scores.
- *Butterflies* and *Poker* modes introduce a more methodical approach to the game, where you must strategize to be able to perform certain actions (destroy butterfly gems or make color combinations) in order to extend the number of plays limit (instead of time limit) - once again focusing on beating high-scores.

Bejeweled has bright colors and animations, combo voice-overs and power-ups as positive reinforcements (dopamine injection), and soothing, ethereal-like background visuals and soundtrack for a chill experience (serotonin injection).

Candy Crush Saga

Come to "revolutionize" the match 3 genre, Candy Crush introduced an extensive level progression system, with different board layouts and challenges for each level. Candy Crush's design assures the player gets engaged:

- Level progression structure lets the player see how much they have progressed and how much content they still have to explore, with a difficulty curve that keeps them feeling challenged.
- Level designs are different, featuring boards of varying sizes and shapes, and different objectives, such as clearing one type of candy, creating and exploding a number of power-up candies, or clearing up obstacles.
- Limiting the number of moves per level makes the player have to think ahead and figure out strategies to complete the goals.

With a more kid-friendly aesthetic, Candy Crush features bright saturated colors and exciting sound effects and animations (serotonin injection) - the haptic feedback from the phone also helps sell the impact of the explosions (dopamine injection).

Spirit Swap: Lofi Beats to Match-3 To **(Demo)**

An indie twist to the genre, Spirit Swap only allows you to swap tiles horizontally, and heavily features Lofi themes, with chill soundtracks and cell-shaded 3d models vibing to the tunes).

- Game pieces keep appearing from the bottom and piling up unless you clear them, and the game ends if any touch the top of the board - tetris style - giving a clear time limit and incentive to act fast.
- Customizable speed and difficulty provides challenging scenarios which test the players' skill.
- Split-screen PVP creates a frenetic competition between two players, highlighting the challenging aspect of Spirit Swap.
- Spells are charged and can be used at your leisure in order to get out of tough spots.

Everything aesthetic about this game is relaxing (serotonin injection) - including environment sounds like birds and wind in the menus - which juxtaposes beautifully with the demanding and sometimes chaotic gameplay, as you rush to

figure out how best to reorganize the tiles before your time runs out (dopamine injection).

Match 3 RPGs Puzzle Breakers; Pirates & Puzzles; Empires & Puzzles; Gemstone Legends; Puzzles & Survival; (...)

At the risk of sounding reductive, these all follow the same formula and will be reviewed as one and the same. The idea is straightforward: take the simple match 3 gameplay and paint it over as a different genre. In these games, you perform actions - such as attacking monsters, building structures, escaping traps - through combining tiles.

- Break up the formula by providing level-specific tasks, such as clearing up specific columns to attack specific targets.
- Character-specific special effects can be fun additions and sometimes have specific activation requirements (such as clearing up tiles of one color), allowing for some strategic engagement.

These games mostly rely on dressing up the simple but tested gameplay as something grander, and can be (and will be, apparently) expanded to any theme - pirates, fantasy adventure, apocalyptic survival, etc. - or even popular IPs.

Games with different *Match-3* mechanics

Some games try to expand upon the fundamental gameplay of matching 3 objects in order to create new experiences. These do not feature the classic tile-swap gameplay, but innovate through different gameplay styles.

- *Zuma* and *Bubble Shooter* are two classic examples which implement the match 3 formula through different mechanics - in their case, shooting balls at same-colored balls. Both of these feature tetris-style limits as the board fills up with balls in time and the player must keep clearing them in order to keep playing and

beat the level or highscore. Both games feature the ability to set up balls in a way that allows the player to clear many in one fell swoop - good strategy mechanic - and focus on a good popping experience.

- *Star Salvager* is a rogue-like spaceship-building match-3 (I-know) game in which you must attach colored tiles to your ship in order to obtain ammo - used to *pew-pew* at enemies. In its current state it is a bit of a mess as the match-3 mechanic while moving and rotating the ship doesn't feel rewarding - no instant gratification - and doesn't mesh well with the progression overall.
- *Matcho* is an upcoming self-called *Match-3 FPS* game where you have to shoot at colored enemies in order to match and defeat them - not much is known about it at the time. Featuring a deeply hurt protagonist and a world-ending event, this bizarre experiment brings more of a reactive gameplay to the match-3 formula, where identifying and aiming at the correct targets fast is key.

These games are a bit out of my scope but might provide interesting ideas, such as a reaction / accuracy based mechanic.

Takeaways

Match 3 games focus on **aesthetics** and **instant rewards** to provide a good and relaxing user experience. How it *feels* to clear tiles is core to this experience – aided by sound effects, animation, scoring and other feedback. **High frame rate** and **responsiveness** are also critical..

Limiting the player can improve the experience. For example, limiting the number of plays can create strategic scenarios, and limiting the time to play can create a frantic sense of having to think – and act – fast. Rewarding the player for clever – or lucky – plays with an extension on these limits provides a sense of accomplishment when beating **highscores**.

Special tiles or **power-ups** are also great ways to reward the player, and add to the satisfaction by allowing them to clear more tiles at once. In addition, little twists on the genre can go a long way to create a fresh experience.

Architecture

MVC

The model-view-controller pattern. Pretty standard stuff. The player interacts with a controller, which in turn informs the model of the inputs given. The model then adjusts accordingly while updating the game state, and sends information to the view, which renders and displays the screen. The screen will then show the player how he can interact. We keep each component separate and they only communicate in this loop.

ECS

Let's not use objects. My object-oriented mind immediately thought of, well, using an object-oriented approach to develop the game, as I am so used to. However, this project posed a great opportunity to finally try a new approach I've been keen on experimenting with: the Entity Component System. ECS mainly tries to fix two problems associated with objects: the problematic scaling of inheritance and its ineffective use of memory.

Now what exactly is the **ECS** approach?

We have **entities**, which can be no more than a simple identifier - an *int* will do.

Entities have **components**, which are structures of attributes - functionality related data.

Systems hold the functionality - they iterate through entities with the required components and do… something.

While researching the topic, I found some potentially useful ECS libraries. In particular, *entt - Gaming meets modern C++ - a fast and reliable entity component system (ECS) and much more.* - seems to be a very solid implementation, and easy to apply to various projects. Although tempting, I did not want to use this library with such a limited knowledge of how an ECS is built, let alone all the other things that it seamlessly implements. *And much more* indeed.

To help me get a grip of how a simple ECS actually works, I based the main functionality of my ECS library on the example provided by Austin Morlan - it is by no means the most complete implementation, but provides a well-structured basis, easy to understand and to iterate upon. I will continue developing this library in order to make some aspects of actual coding more accessible, such as a better way to register systems with their respective signature.

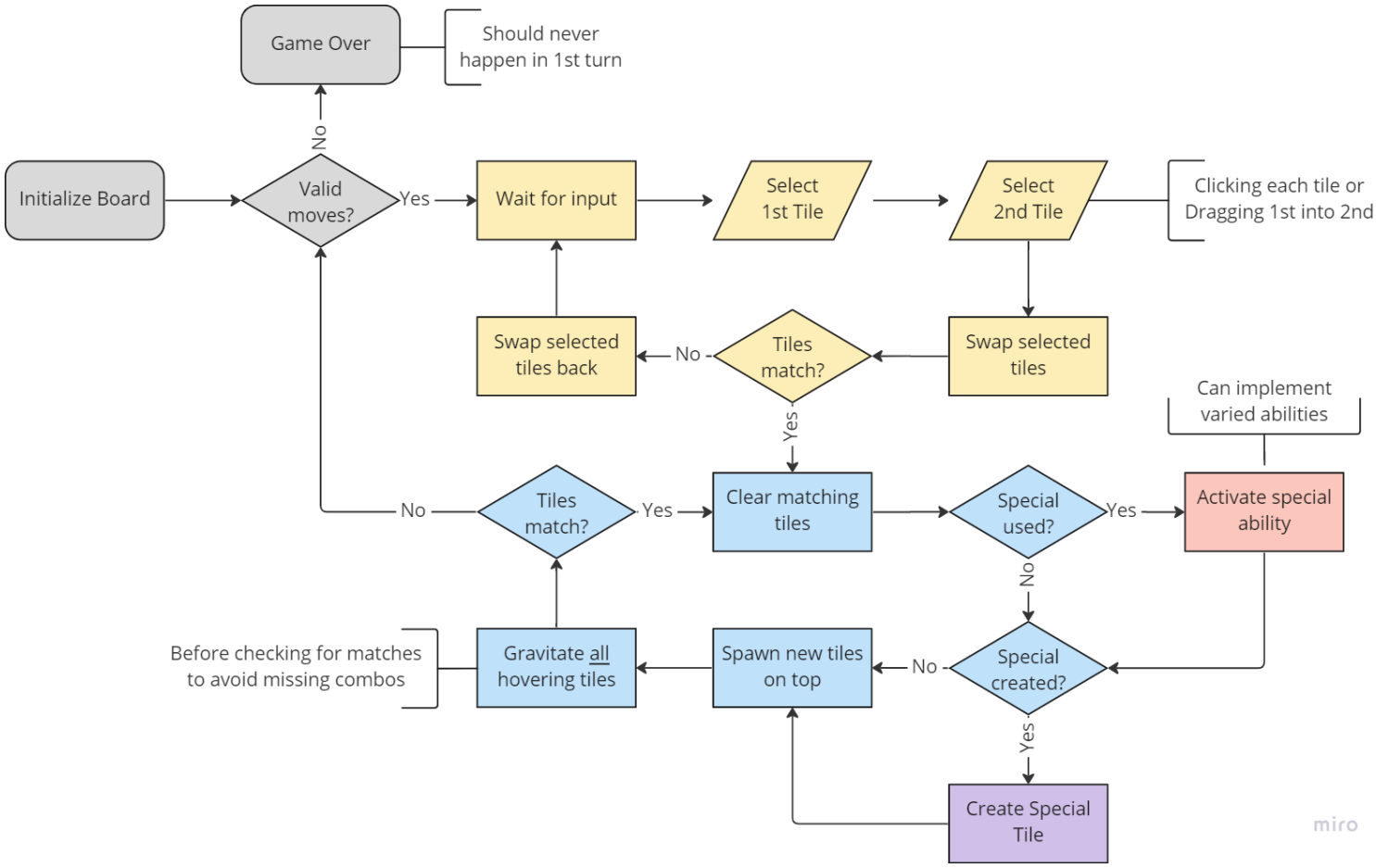
Game Design

Core Loop

A pretty simple design. We initialize the board and then have a main loop composed of two smaller loops:

1. An input loop (yellow) where the player searches for a valid match;
2. A clearing loop (blue) where matching tiles are destroyed, new tiles are spawned accordingly, then all tiles are gravitated to fill the board, and new matches are once again cleared.

Once these two loops are completed, we check if there are valid moves left and start over. For different game modes, we might not want to end the game if there are no more valid moves - and instead shuffle the board for example. We might also want to add new mechanics such as activated abilities - expanding the input loop. Spawning special tiles and activating tile abilities are neatly inserted into the clearing loop.



Implementation

This is why we build pretty diagrams, our code implementation is going to follow it pretty closely. We'll have a main gameplay loop – *while (running)* – with two loops inside - *while (no_matches)*, which loops until the player makes a valid move, and *while (not_stable)*, which loops until the board has once again achieved a stable state (no matches currently on the board).

We will have a Game class acting as an Engine, and GameScenes with States – rectangles in the above diagram – which will update the game – following a Model-View-Controller design.

The bulk of the functionality will be handled by systems, of course. We'll have engine-wide systems such as a Render system (which will render all renderable entities) and gameplay-specific systems, such as MoveTileObject system. Our board – which will have a grid (vector) of entities – will act as a pseudo-system, handling the creation of new entities and adding components to them.

We must also register appropriate components for our systems to use. A standard Tile entity will have components such as *sprite* (to be renderable), *position* (so we know where to render it), and *tileObject* (containing *color* and *movable*). We might then create immoveable tiles, or tiles with no color - which can be moved but never cleared through matches. We can also create special tiles with a *tileAbility* component which contains what kind of ability it activates.

main() → calls Game.Run().

Game → initializes SDL, creates initial GameScene, handles main loop (timer and updates per frame), calls GameScene to handle gameplay loop.

GameScene → sets up scene resources, registers components and systems, handles gameplay loop through MVC pattern:

- GameScene.HandleEvents() → controller
- GameScene.Update(TIME_STEP) → model
- GameScene.Render() → view

Level : GameScene → creates Board, contains LevelStates

Board → generates a grid of TileObject entities, handles logic like:

- turning screen position into grid coordinates
- checking for matches
- flagging TileObject entities for destruction by adding a component
- updating grid after moving / deleting / spawning TileObjects

User Experience

Aesthetics

This type of casual puzzle games thrive when focusing on some kind of ***chill*** aesthetic – we've got *Bejeweled's* ethereal ambience, *Candy Crush*'s cheerful setting, and *Spirit Swap*'s laid back lo-fi vibes. I set out looking for my own aesthetic with a few requirements in mind.

- First, I want to create a simple, catchy soundtrack - one that can sooth the player while they play, help them take their mind off things (and preferably one that would be easily loopable).
- Next, I want good bouncy popping sound effects – to really nail home those good sensations of clearing a set of tiles. *BOP*. Something percursive but melodic.
- Lastly, I want to neatly tie the audio with colorful visuals – a soothing backdrop, accented by the inevitably chaotic nature of clearing tiles and dropping more tiles to be cleared through bright animations.

I need something cool . . . something groovy . . . something . . . jazzy. *Jazz*.

Arguably I'm getting myself out of my depth here but something just clicked when I thought *jazz*. My idea is to heavily feature those warm yellows, pinks, purples, usually associated with jazz – I think it will make a good color palette for our tiles – and play a soothing jazz piece for background music. The sound effects: jazz instrument sounds.

Maybe a drum set for when we clear different colored pieces: a cymbal for yellow and a snare for pinks? Combo and power-up sounds can be played by a bass, a sax or a piano. The real quicker: the sound effects play in time with the soundtrack. Whenever the player clears a set of tiles, the percussive sound would pop up at the next appropriate time, adding seamlessly to the soundtrack while providing audio feedback about their move. An example of a similar mechanic is found in *Zelda: Breath of the Wild* : when fighting enemies, musical accents are added to the soundtrack whenever you hit an enemy (at the appropriate musical time). Now, this probably isn't very feasible in this little time, but I still want to prototype a version of this system. And even if I can't implement something like this so fast, I'll still have jazz.