JULY 3, 2023

# REPORT

MUHAMMAD TAHA

# TABLE OF CONTENTS


Introduction ..... 2
Action Taken ..... 2
- Memory Footprint ..... 2
- Framerate ..... 2
- Garbage Collection ..... 3
- Binary Size ..... 3
- CPU Cost ..... 3
- Editor Compilation ..... 4
- Project size ..... 4

Achitecture ..... 4
- Singleton ..... 4
- Scriptable Object Events ..... 4
- SOLID ..... 5

Noticeable Bugs And Modifications ..... 5
Extra Development Techniques ..... 5
- Pooling System ..... 6
- Editor Coding ..... 6
- Helper Classes ..... 6

Fun Factor ..... 6
Art References ..... 7
Level Design ..... 7

# INTRODUCTION

This submission include Android APK, report and project developed on Unity 2021.3.12f1.

I tested build on my android device Samsung S22 with OS 13. I tested it on other devices as well, like Redmi Note 11 with OS 11 and Redmi Note 10 Pro with OS 13.

According to the tasks, actions taken for:

- Memory Footprint.
- Framerate.
- Garbage collection.
- Binary size.

Besides given tasks, I further take more actions for optimizing:

- CPU cost.
- Editor Compilation.
- Project size.

Architecture I used in the project is hybrid.

1. Singleton Pattern.
2. Scriptable Object events.
3. SOLID.

In the last, I will explain few notable bugs and their improvements. Also, some extra positive development techniques and fun factors. I will also give references for the free-art assets I used as I am not an artist. I will also pass some information about the level design as well.

# ACTION TAKEN

As given tasks, actions taken for every task is explained below:

## MEMORY FOOTPRINT

To optimize memory footprint, I used pooling system for spawning and despawning tetris tiles and particles. I try to not use extra variables in the loops or in update function to avoid caching.

I am using Singleton pattern for managers, so I try to not put them in the scene. That way the information of objects also not cached in the memory. This also helped to optimize the memory and garbage collection as I don't have to destroy them.

## FRAMERATE

To optimize frame rate, I had to minimize the use of physics, update functions, rendering, audio and code stripping.

To optimize **physics**, I crosschecked the "Reuse Collision Callbacks" to true. Then, I enable experimental multithreading option and minimize the fixed timestap to 0.03.

To optimize **update functions**, I removed empty update function from every class. I try to not use update and switch most of the code in the coroutine with some delay, this is way much better than the update. I only use Update function in the InputManager, Player and Meter Update.

To optimize **rendering**, I modify few setting in the player setting, like multithreading rendering, graphic jobs and batching with the Scriptable Render Pipeline (SRP). In SRP, I modify few things like shadows, lights, anti-aliasing and disable HDR.

To optimize **audio**, I modify few settings in the project setting like DSP buffer and then enable "force to mono" while importing.

To optimize **code stripping**, I enable that in the player setting to remove extra code from the build time.

## GARBAGE COLLECTION

For garbage collection, I try not to use logs and used pooling so spawning and destroying won't happen. Then I enable the incremental GC from the player setting.

I also didn't use "new Wait for seconds" where ever possible, also in collision/trigger callbacks I use "CompareTag" function instead of "tag" to avoid creating extra variables.

## BINARY SIZE

To optimize binary size, I compress textures on ASTC 6x6. I compress audio as well like quality to 50 instead of 100. To make it more optimized, we can use assetbundles and addressables.

I also disable few extra unity built-in libraries from package manager like AI, navmesh and many more. The reason to do that, those libraries won't be added in the build, also optimize cpu cost and Editor Compilation.

## CPU COST

To optimize CPU cost, I try to use coroutines as they are multithreaded and good substitute for Update function (but they are not pure obviously, for that I had to use Job system which is extra overhead for current scope).

I try not to create new variables at runtime, and try not to use new vector too much, instead using static vectors like vector3.one etc and modify them for my need.

## EDITOR COMPILATION

To optimize editor compilation I organize code properly, and create assembly definition. This way only the modified assemblies will compile. If we don't create assembly definition then, all code will be gone to Unity assembly and compile full assembly when changes are made.

## PROJECT SIZE

I removed extra files from the project whether they are scripts, images or anything. I didn't use PSD file as a sprite, instead I convert them in PNG, as PSD files have more size.

# ACHITECTURE

I use hybrid architecture to make sure that my code stays clean and expandable code.

## SINGLETON

I use most managers as a singleton, and most of them spawned when first call is made. As an example you can check Audio manager, Save progress and Game Manager. I specifically give these three as an example because all of them have different characteristics.

Audio Manager spawn from the resources, Save Progress create object at runtime and Game Manager doesn't even create an object in hierarchy.

The reason to use different approach is, because Audio Manager have references from the project, like audio clip, audio sources and custom variables. So, doing them via code is complex approach and difficult for the Game designers to modify as well, as these areas should be open for modifications.

Now about the Save Progress, I created object at runtime because nothing is linked from the project or hierarchy, and solely based on the code.

Now about the Game Manager, I usually use Game Manager to communicate between scenes. In my case background art should be consistent among the scenes. To handle that efficiently, Game Manager came in help.

## SCRIPTABLE OBJECT EVENTS

I try not to use delegates or actions, as they need more configuration of the code. That doesn't mean I didn't use them, they are very handy. But scriptable object are really handy for the game design, as game designer can't write code, so SO-Event came handy there.

I even use Actions in the Input Manager to tell about the swiped made. I used there because whoever had to modify input swipes code, they will probably be a developer not the game designers.

## SOLID

Most of the code, I try to write is open for editing via events or callbacks. This is really helpful for the game designers to add more functionality.

I even try to do single responsibility to one script, like tile behavior only responsible for all kind of tile behavior we have right now. If we need to add different variation of the behavior then we can take the example of the User and AI class. Both are inherited from the player class and tile behavior cast them to the player class, in this way tile behavior doesn't need to know whether it is AI or User. So, adding different tile behaviors, I will have to change code to inheritance.

# NOTICEABLE BUGS AND MODIFICATIONS

I have a checklist in my mind to add in the game if more time was given.

1. I see one bug that tile collision corner isn't smooth, like sometime it seems it will place there correctly but fall in acid. See figure below.


2. To optimize binary size more, I can implement assetbundles to download related assets at runtime.
3. I have to add confetti particles at level complete to encourage users with a win.
4. Will have to add haptics for iOS, Android is there but missing for iOS.

# EXTRA DEVELOPMENT TECHNIQUES

I use few extra techniques to handle project more efficiently. These are mentioned below:

- Pooling System
- Editor Coding

- Helper classes

## POOLING SYSTEM

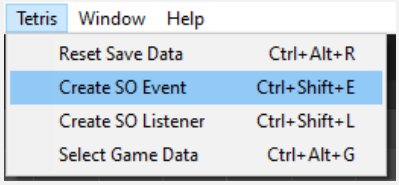
I used pooling for the tetris tile and particles. This optimize a lot of things like memory, cpu cost and stable framerate.

I even use Dictionary there for finding a right list of objects instead of iterating them in loop, this is also important for the optimizing the search cost among the pool.

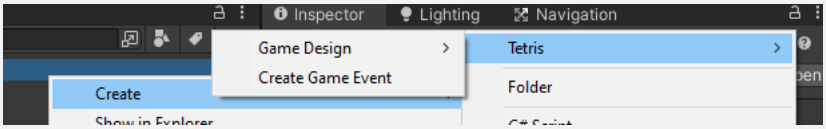
## EDITOR CODING

I write few editor code for the utility, like menu items, creating new level scriptable object and many more. In the editor you can find them in the "Tetris" tab.

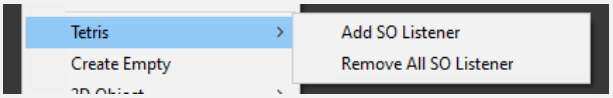
Toolbar item like reset save data, select game data and few SO Events are created. All of them have short keys like control+alt+r, control+alt+g and control+shift+e and l respectively.




Right click on the project window to create new SO's for the level, game data and Events.




Lastly right click on the hierarchy to add or remove SO-Listeners to selected object.




## HELPER CLASSES

I write few helper classes for the developers which act separately and can be used in any project. Like camera shake, Rotate in coroutine and few maths functions.

## FUN FACTOR

I added a lot of fun factor in the game, to make it more interesting to play.

- In the game initially required tower height is less and it increases by level to make user to play more and more.

- I added camera shake when tile hit the ground.
- I added haptic feedback when tile hit ground as well as when fall in the acid.
- I added simple particle effect when tile reach the ground or hit acid.
- I added UI animation to make it feel more live.
- I added height meter to tell user about the progress, and it changes color base on height. Red, yellow and green.
- I added extra input feature in the game as well, like swipe down to make it fall quick and swipe up to reduce speed.
- I added instruction in the game to explain game to user.

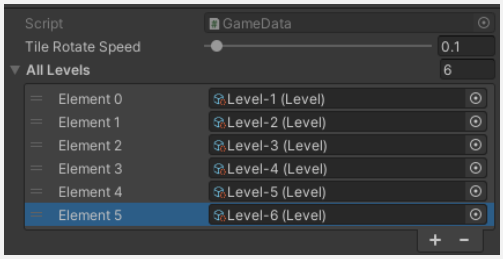
## ART REFERENCES

Here are the art references:

- Visual Art:
  - Gameplay background assets used from GameKit 2D.
  - UI is downloaded from here, then sliced from the photoshop.
  - Logo is used from here, and modify from the photoshop.
  - Tetris tiles are used from here, and sliced them from the photoshop and fix w.r.t game play. Like 1 small cube in the game is 256 pixels, so 2x2 tetris tile should be 512x512.
- Audios:
  - Click audio clips used from here.
  - Impacts audio clips used from Kenny Impacts pack
  - Background music used from here.

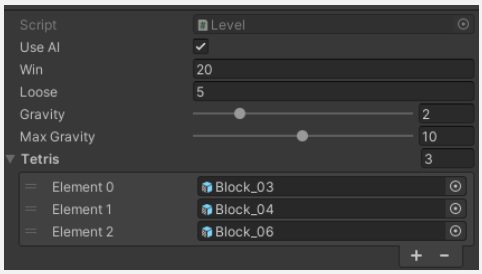
## LEVEL DESIGN

Currently there are 6 levels which rotate when all are completed, you can see them in the scriptable object folder or you can select game data by hitting control+alt+g. In the game data you see 2 varaibles.




1. **Tile rotate speed**: This variable handle the rotation speed of the tile when tap on the screen.
2. **All Levels**: This array contains all the levels. You can select any of it.

In level you can see bunch of variables.




1. **Use AI**: You can enable/disable AI for specific level.
2. **Win**: You can give the max height of tower for this level.
3. **Loose**: You can give the amount of tiles falls in the acid.
4. **Gravity**: Default fall speed of the tile.
5. **Max Gravity**: When swiped down how quick the tile fall, or new gravity.
6. **Tetris**: List contains all tetris tiles to come in this level.

These bunch of variables can create a lot of variations in the level.