Puzzle Game Engine

Thank you for purchasing „Puzzle Game Engine”, I hope you will enjoy using it. If you have any questions or issues, please don’t hesitate to email me:

ragendom@gmail.com

CONTENTS

1. Changelog

V1.0

Added 8 Games:

#1 Sort Hexagons, #2 Sort Cards, #3 Sort Coins, #4 Sort Cake, #6 Sort Cards 2, #6 Sort Cubes, #7 Sort Nuts, #8 Sort Coins 2

V2.0

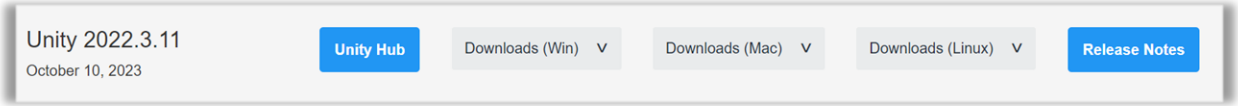
Added 5 Games:

#9 Unpuzzle, #10 Sort Dominos, #11 Tap Away 3D, #12 Unscrew Jam, #13 Bottle Jam

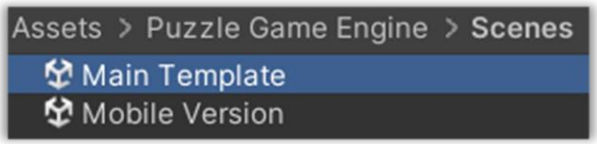
2. Setup

CLICK HERE FOR VIDEO TUTORIAL

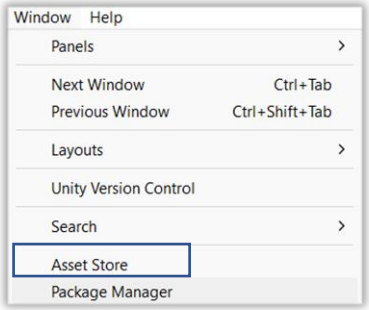
- **Use 2022.3.11 version of Unity** for the project to avoid issues caused by version difference

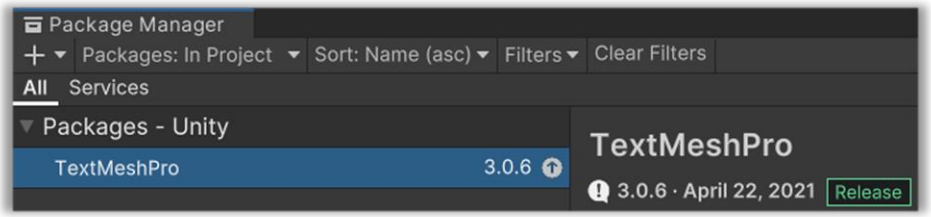


- **Drag&Drop** the **.unitypackage** file to your „Assets" folder inside „Projects", or go to „**Assets**" window -> „**Import New Asset**" -> then select the **.unitypackage** file
- Open **„Main Template"** scene (***Assets -> Puzzle Game Engine -> Scenes)***



- Import **„TextMeshPro"** package to see texts (***Window -> Package Manager -> Search for TextMeshPro)***





3. Creating Levels

CLICK HERE FOR VIDEO TUTORIAL

4. Project Guide

Hierarchy Structure

The project has 3 main areas, each have playable templates inside:

1. **Stacking Games**

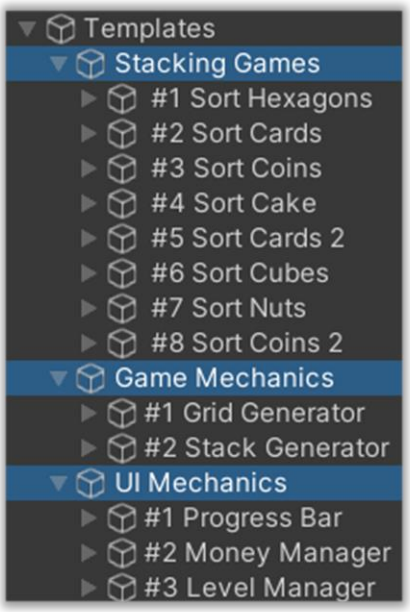
You can see what kind of games are possible to build with this engine. Play and inspect how these game templates handle each system and mechanics to create variety.

2. **Game Mechanics**

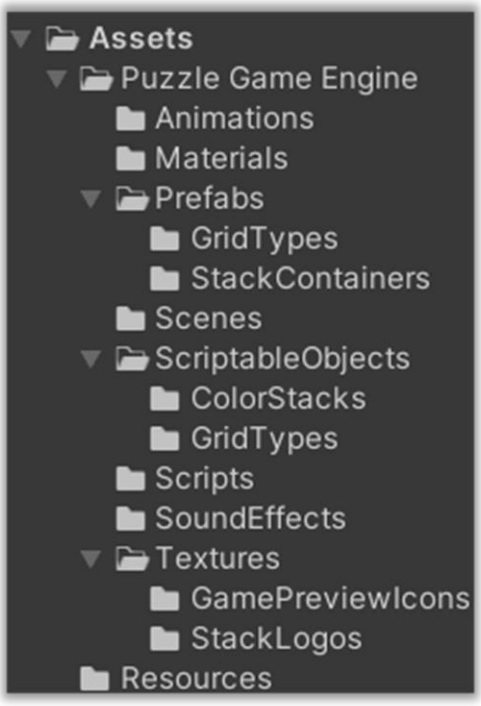
The main game systems used to create the games. You can study how each mechanic is used in an interactive way.

3. **UI Mechanics**

The main UI systems used to create the games. You can study how each mechanic is used in an interactive way.



Assets Structure



- **Puzzle Game Engine**

Contains all of the assets folders, except for Resources

- **Prefabs**

GridTypes and Stack containers contain prefabs which are referenced to grid and stack spawners. Most of the prefabs have different values such as spawn properties, meshes, etc.

- **Scriptable Objects**

ColorStacks are used to determine which colors can be choosed. You can modify the colors and count of colors to make the stack spawner decide how many and which colors can be choosen.

GridTypes are used to choose which type of pre-defined prefab is spawned if not specific prefab is selected at the spawner script. (Stack Container Random Spawn.cs)

- **Sound Effects**

Contains .mp3 files for sound effects. Each template has a script called „SoundsManagerForTemplate.cs”, which decides which sound effect is played in which case. You can assign your preferred sounds in script’s public variables in the Inspector.

- **Textures**

GamePreviewIcons are textures which are used in the scene called „Mobile Version”. The game navigates to the selected game based ont he preview icon.

- **Resources**

Contains a prefab which is spawned by the script „ParabolicJump.cs” at the end of a stack piece’s jump. It is spawned because it is a trigger which sets the piece to the final position once collides with it.

5. Scripts

ActiveStackContainersRandomManualRespawn.cs

Spawns objects at currently empty grids. Count of objects to spawn is random between a minimum and maximum amount. Can spawn at start of the game, and can be called to re-spawn later. Can have a minimum count of empty grids to spawn.
Usecase: Game #8

ActiveStackContainersRespawn.cs

Respawns objects if a container is empty or if all containers are empty. It can be used at dealing mechanic as well, it is pre-spawned under the ground so player cannot see it.
Usecase: Game #1

AnimationPlayer.cs

Checks if object has the requested animation clip and plays it if the object does have it.
Usecase: Game #2

CheckNeighbours.cs

Checks if a grid's stack pieces can jump to another grid. Manages the jumping mechanic, matching of the stack pieces, re-checking neighbour grids, etc.
Usecase: Every game which have a grid system

CheckNeighboursQueue.cs

Makes sure that at a time there is only 1 grid checking their neighbours for match. Handles a queue to achieve it, grids can be queued and enqueued to check neighbours for matching.
Usecase: Game #1

Clickable.cs

If object is clicked, then it tries to jump to a grid. Jumps if it finds a matching top color or if it finds an empty grid.
Usecase: Game #5

CollectedEffects.cs

Keeps a pool of objects, enables and disables them when its function is called.
Usecase: UI Mechanics #3

CollectedPiecesCounter.cs

Handles money collection. Updates UI with correct collected count, saves and loads collected count, plays animations on collect event.
Usecase: UI Mechanics #3

CollectedStacksCounter.cs

Keeps track of how many object matches were in the game. If it reaches the needed amount per session, then an OnCollectedAll event is invoked. It is used to determine if a level is completed.
Usecase: Game #5

ColorManager.cs

Manages the color selection of object. Gives back its current color, sets new color, resets material to default color. Also handles visibility of object before choosing color. StackColors scriptable object is used to choose colors from.
Usecase: In every game

CreateNewVisualAtStart.cs

Grids can spawn a new look instead of using their MeshRenderer. The spawned object need to follow the grid's position, that's why it has the FollowObject.cs script. Typically used in games where the grid is a complex mesh with sub-meshes.
Usecase: Game #3

DealStackButton.cs

Used for dealing cards and coins to grids by player. It can have a 3d object as button or a 2d UI button.
Usecase: Game #2

DestroyGameobject.cs

Destroys the gameobject which has this script attached to. There are events which are invoked at OnDestroy. Manages enabling and disabling other gameobjects OnDestroy.
Usecase: Game #6

Draggable.cs

User can drag this gameobject. Typically used to drag around stacks and to place them on grids. Handles the grid's highlight while dragging, handles moving the draggable gameobject, placing object to selected grid, moving it back to its original position if no grid is selected. Decides if object can be dragged only once or countless times.
Usecase: Game #8

FocusableObject.cs

Controls the main camera's focus. Main camera will have CamHolder gameobject as parent when reaching near this FocusableObject, and will have it's position and rotation.
Usecase: In every game and mechanics template

FocusableObject.cs

Controls the main camera's focus. Main camera will have CamHolder gameobject as parent when reaching near this FocusableObject, and will have it's position and rotation.
Usecase: In every game and mechanics template

FollowObject.cs

Continously follows a gameobject with an added offset. Typically grid's visual placeholders have this script attached to them to match the grid's position.
Usecase: Game #3

GridChildrenAreSelectable.cs

Levitates the selected grid's stack pieces.
Usecase: Game #2

GridGenerator.cs

Manages everything regarding to spawning grids. You can spawn in linear or circular pattern with even distribution. You can spawn different shaped grids, for example hexagon, square, coin, rectangle.
Usecase: Game #1

GridTypes.cs

Scriptable object, defines the available grid types to choose from. GridGenerator.cs can choose from the defined grid types to spawn.
Usecase: GridGenerator.cs

HideColor.cs

Enables or disables the hidden color gameobject. Used to add a game mechanic where the color of the stack's next object will be shown only after the previous object is moved.
Usecase: Game #7

HighlightMatchingStack.cs

Used to highlight a whole stack of objects, which are full and matching. Shows the player that the stack is ready to be merged together.
Usecase: Game #3

LevelManager.cs

Manages saving, loading of levels. Activates and deactivates panels such as level cleared and level failed panel.
Usecase: Game #5

LockedSlotsWithKey.cs

Adds a game mechanic where the grid's slots are locked. Manages locking and unlocking the slots, spawning the locked visual prefab with offset.

Usecase: Game #6

LockInStack.cs

Used to lock a grid / all slots of a grid, once the needed amount of matching stack objects are added to the grid. Has events to invoke on grid lock, can animate the stack when locking and play particles. Prevents player to drag to this stack any further or to move its elements.

Usecase: Game #6

MainCameraController.cs

Manages player inputs. Moves, scales and rotates the main camera based on player inputs. Automatically focuses on the closest game template

Usecase: On player input

MatchCompletedEffectSpawner.cs

Used to spawn an effect when the cake slices are matched

Usecase: Game #4

MatchSymbolWithColor.cs

Used to select symbols (sprite) of object based on the object's color.
Usecase: Game #6

MergeStackButton.cs

Used to try to merge stack objects in all of the game's grids.
Usecase: Game #3

OccupiedGrid.cs

Can set a grid occupied and can choose if dealing to this this grid is possible or not. Spawns an occupied prefab to this grid
Usecase: Game #2

PagesSystem.cs

Used to showcase UI Mechanics in an easier and space-efficient way. Makes it possible to switch pages so that we can show more elements easier where only one page is visible at a time.
Usecase: UI Mechanics #1

ParabolicJump.cs

Typically the object has this component which needs to be stacked. Manages the jumping/moving mechanic of the object to a grid. Re-checks grid neighbours after object is jumped. Defines different jumping mechanics (for example parabolic jump, or simple linear movements)
Usecase: In every game

PhysicalButton.cs

Physical button with pressing mechanics and OnButtonPressed events to invoke. Has an option to press button only if has enough value to press it. Typically used for unlocking Game #6's locked stack slots with a key.
Usecase: Game #6

PositionAnimation.cs

Animates position changes via script instead of animation. Main benefit is to be able to animate position changes relative to start position (cannot do that with generic animation)
Usecase: Game #8

ProgressBar.cs

Used to track how much percentage of the stacked objects have a grid relative to how many objects are needed to match.
Manages everything regarding a progress bar, for example: different speeds, different colors, events (start, change, finish)
Usecase: UI Mechanics #1

RandomizeChildrenPosition.cs

Randomizes position of children relative to their start position. Chooses a random amount between minimum and maximum position offsets. Also able to reset to the original position

Usecase: Games #5

RandomizeChildrenRotation.cs

Randomizes rotation of children relative to their start rotation. Chooses a random amount between minimum and maximum rotation values. Also able to reset to the original rotation

Usecase: Games #5

Rotate.cs

Player can rotate the gameobject on Y axis by moving swiping left-right. Rotation can be snapped to closest rotation amount

Usecase: Games #1

RotateAndMoveAround.cs

Used to rotate and move around continously the gameobject once it is selected. Smooth movements and rotation is possible without animation. Randomizes the rotation and position.

Usecase: Games #6

RotateConstantly.cs

Used to rotate the object around any axis. In Game #7 it is used to rotate on and off the nuts from the cylinder while the object is jumping to other cylinder and on selection (by rotating on Y axis).
Usecase: Games #7

RotationAnimation.cs

Animates rotation changes via script instead of animation. Main benefit is to be able to animate rotation changes relative to start rotation (cannot do that with generic animation)
Usecase: Games #5

ScaleAnimation.cs

Animates scale changes via script instead of animation. Main benefit is to be able to animate scale changes relative to start scale (cannot do that with generic animation)
Usecase: Games #4

ScaleDownAndDestroy.cs

In Game #4 it is added tot he matched pies runtime by CheckNeighbours.cs so that the pies are scaled down and then destroyed.
Usecase: Games #4

SelectableAfterPlaced.cs

In Games #6 and #7, it is used to be able to select the object even after it is placed to a grid. Player can re-select it and place to another grid. Manages the levitating animation (speed and offset) and outline while object is selected.
Usecase: Games #6

ShowcaseParent.cs

This script is added to every template, other scripts can access all of the current game's variables via this script. It helps to differentiate the templates from each others. Also saving and loading (playerprefs) uses the object's name which has this script. Also keeps track of levitating objects, colors, currently selected grid stacks, jumping stack, etc (to make sure that at one time only 1 stack can be interacted)
Usecase: Every template

Slice.cs

Used only in Game #4. It is essential to use this with sorting cakes, because that game has entirely different stacking hierarchy than the other games. This script is managing the movement, neighbour checking, disabling of slices.
Usecase: Game #4

SlicesOrderNextToEachOther.cs

Used only in Game #4. Makes sure that the slices are placed next to each other ont he plate. Leaves no empty slice between 2 slices
Usecase: Game #4

SpaceChildrenEvenly.cs

Spaces children evenly on each axis. Can center pivot to transform.
Usecase: Game #2

SpawnerOfStackContainer.cs

Spawns stack containers, chooses animation to play after the objects are spawned. Can choose between manual and random spawner prefabs
Usecase: Game #8

SphereCaster.cs

Typically grids and draggable objects have this script attached to them. It casts invisible spheres which can collide with grids most of the time. It then checks of grid is empty or not (so we can drag object to grid or check grid neighbours for matching colors for example). Can choose various casting properties, such as: which layer it collides with, cast size and distance, cast direction.
Usecase: Game #8

StackColors.cs

Scriptable object, contains all of the possible colors to choose from. ColorManager chooses colors from this list. It can return a random color from the list.

Usecase: Every game

StackContainer.cs

Manages destroying, spawning, coloring of stacks. Can select various spawn properties, such as spawnable object's type, distance of spawned objects.

Usecase: Game #1

StackContainerRandomSpawn.cs

Similar to StackContainer.cs, with added variables regarding the randomization of the spawn. Can set the minimum and maximum amount to spawn, minimum consequent colors next to each other, how many maximum colors should one stack have, etc..

Usecase: Game #6

StackContainerRandomSpawnSlices.cs

Similar to StackContainerRandomSpawn.cs, but made explicitly to handle pie spawning only. Only used in Game #4

Usecase: Game #4

UnlockableGrid.cs

Manages the buying and unlocking process of locked grids. Has events, such as OnPurchase and OnFailedPurchase. Can choose a color to display when player has enough money to buy the grid. Can spawn locked gameobject visual to grid. Grid cannot have stacks while it is locked.
Usecase: Game #2

UnlockedColors.cs

Defines which colors are unlocked at start. Uses Playerprefs to save and load unlocked colors per session. Gets next unlockable color from Stack Colors scriptable object. Can unlock colors (other script calls its function, usually after matching stacks)
Usecase: Game #3

UnparentOnEnable.cs

Used only in Game #6 for unparenting EffectHolder when enabled (usecase is related to unlocking locked slots of grid)
Usecase: Game #6