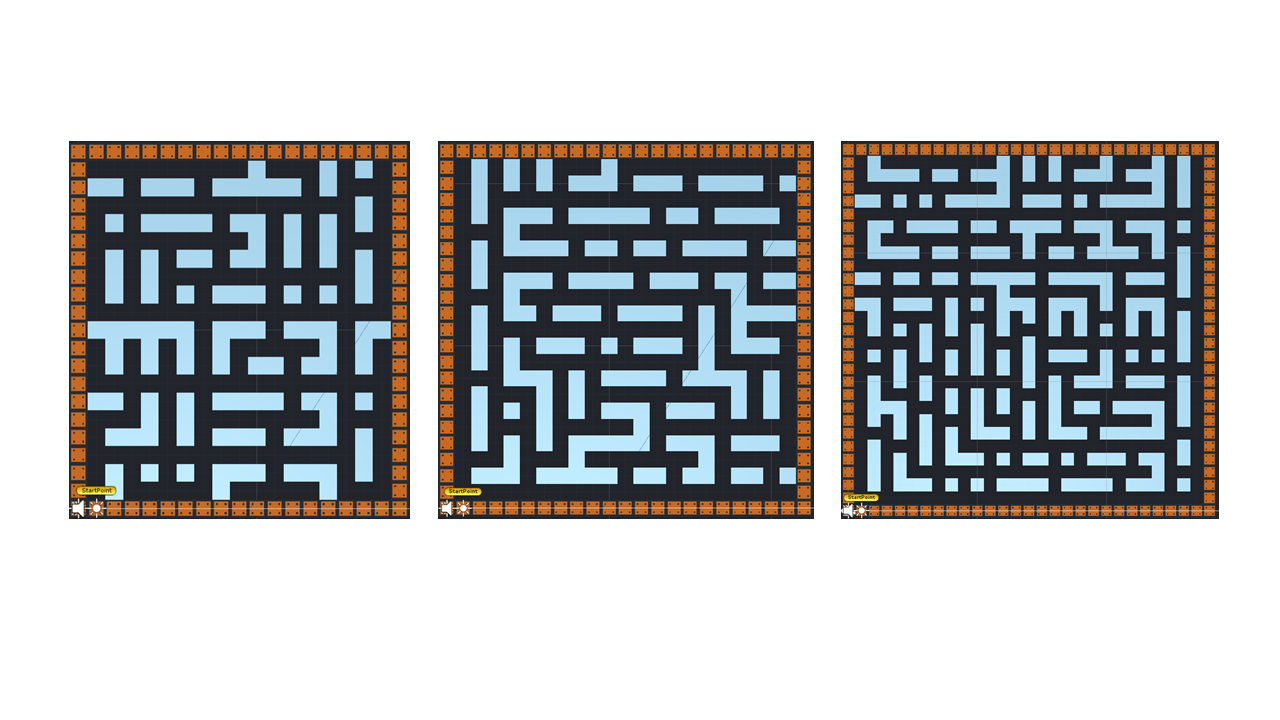
1. **Realisation**

In the implement stage, most game objects meet the requirements of design and all expect functions are realised in the final product. Some modifications have also been made to the ‘Item’ game object to make it fit requirements of this game better. Some extra components including animation system and particle system are also added to provide better gameplay experience. This section will discuss the implement for each component in detail.

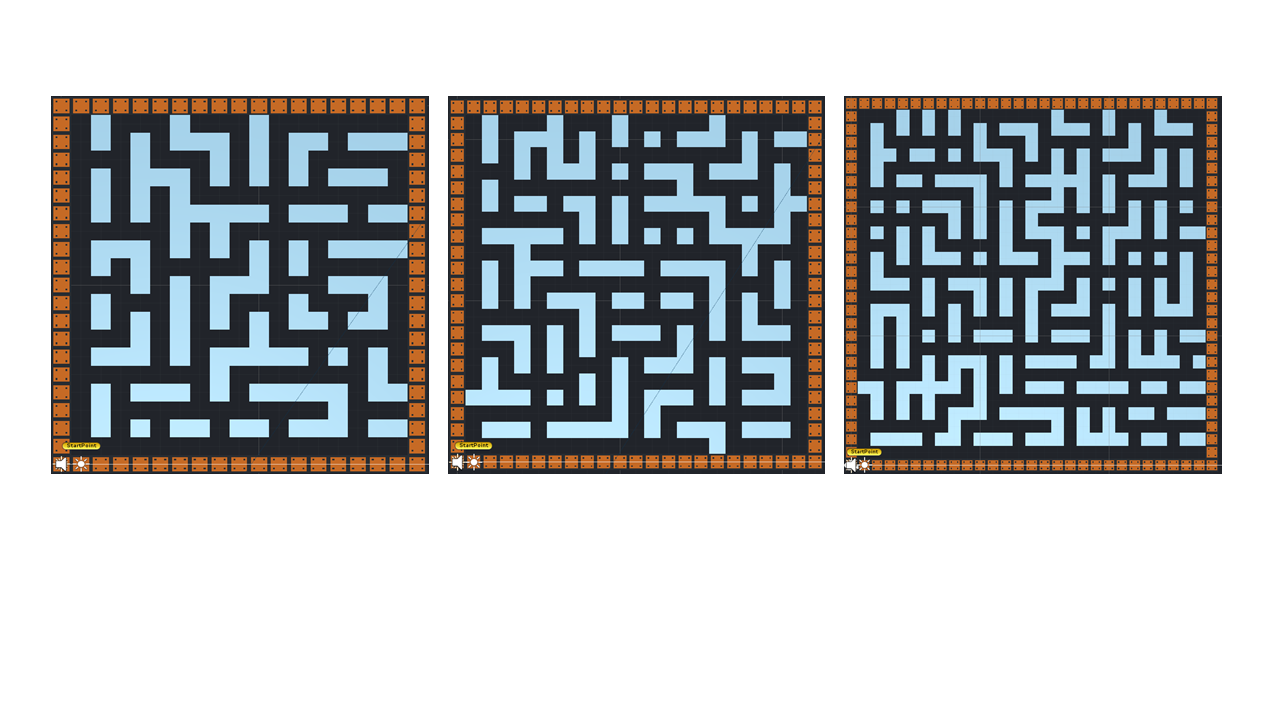
* 1. **Maze Generation**

Since the maze generation system is the basic and key component for the game, it should be completed first. All three algorithms mentioned in the ‘Design’ section are implemented in the Unity environment. An abstract super class “MazeGenerator” is created as the parent of all maze generators. The MazeGenerator class also contains necessary API for the generation of mazes. Then, three C# scripts are created to implement the logic of each maze generation algorithm. Finally, an empty GameObject is created in the game scene and three MazeGenerator scripts are attached to that object. When *MazeGenerator.GenerateMaze(width, height)* function is called, mazes with different types and sizes will be generated. Figure 6.1.1, 6.1.2 and 6.1.3 shows mazes generated by different algorithms. Complete codes for maze generation can be found in appendices.



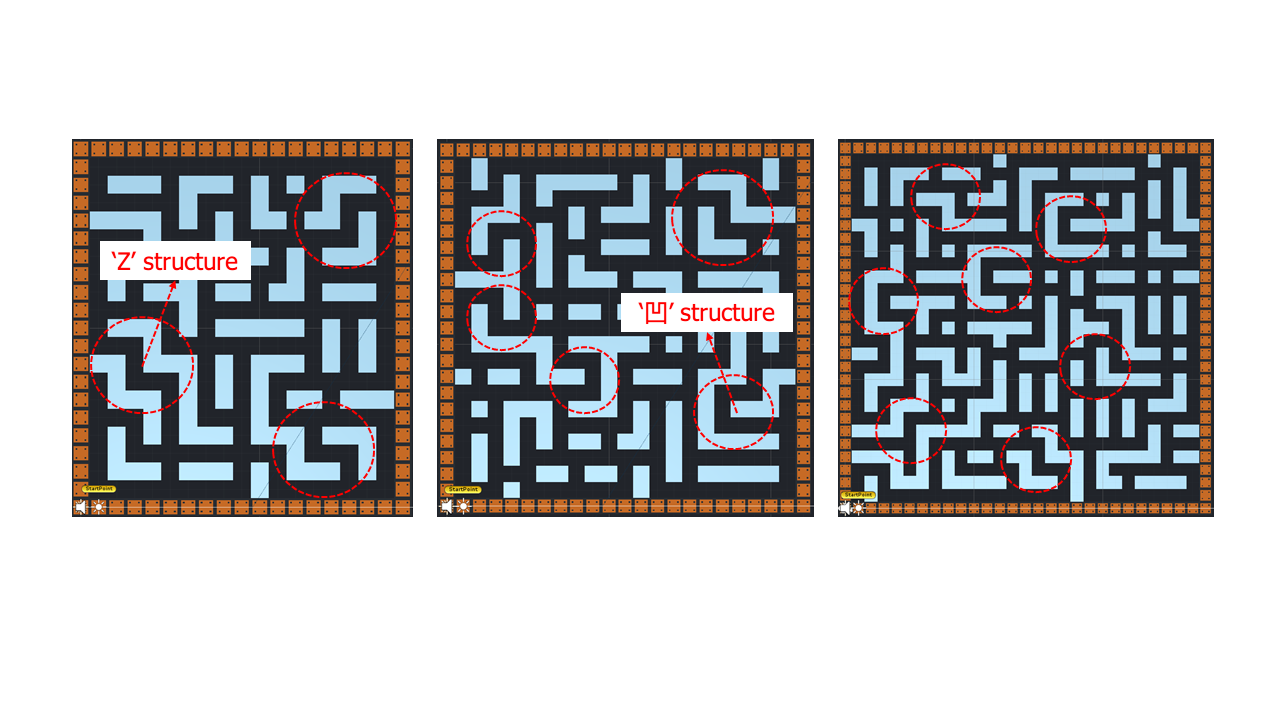
*Figure 6.1.1 Mazes generated by recursive division*

Mazes in Figure 6.1.1 are generated by recursive division. Mazes generated by this kind of algorithm usually consist of many short vertical and horizontal walls. There are also many long, straight roads, which makes this maze easy to solve. Therefore, this kind of maze generator are used to create the simplest maze in the game.



*Figure 6.1.2 Mazes generated by randomized Prim’s algorithm*

These mazes shown in Figure 6.1.2 make use of the randomized Prim’s algorithm. Compared with recursive division, this algorithm can generate mazes which have more corners. Players need to adjust PacMan’s direction more frequently and try to find different way to solve the maze. This maze generator is used to provide mazes with medium difficulty.

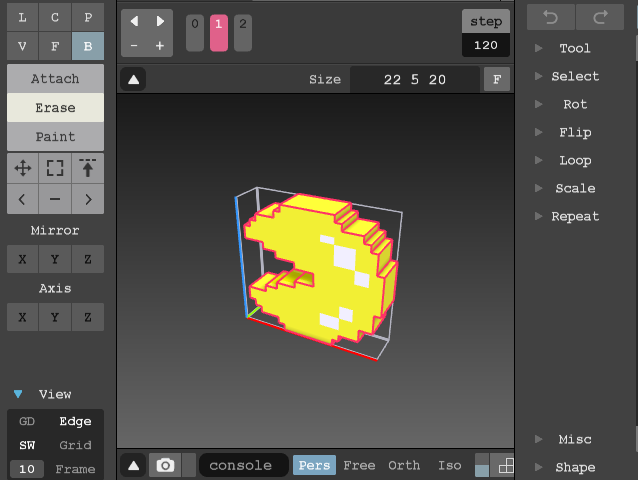


*Figure 6.1.3 Mazes generated by recursive backtracker algorithm*

Mazes generated by recursive backtracker algorithm contain many ‘Z’ and ‘凹’ structure (Figure 6.1.3). Both ‘Z’ and ‘凹’ structure have a feature that if a PacMan enter this structure, there will be only one exit. In this game, ghosts will be placed in the maze to hunt PacMan. Therefore, such kind of structure can be dangerous because PacMan will have no other ways to run. With this feature, recursive backtracker algorithm is used to generate those most difficult mazes.

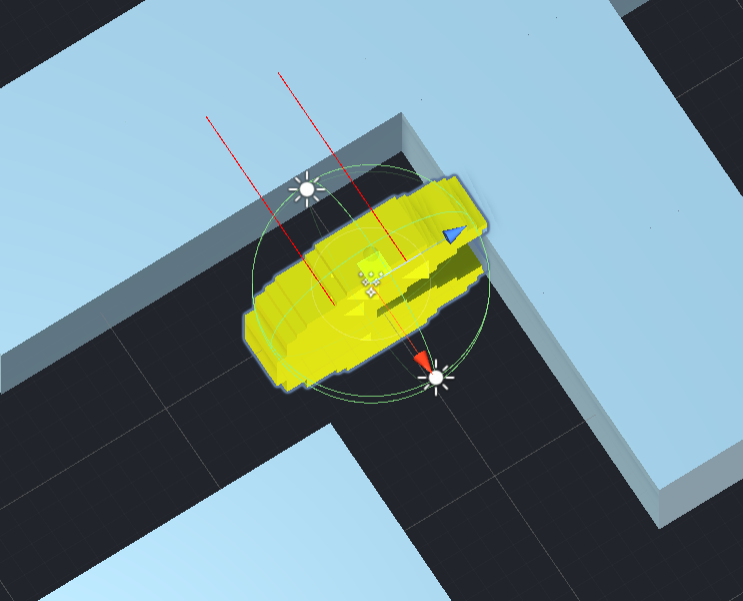
* 1. **Player**

Player (PacMan) is implemented according to the design document. Pseudo codes for the logic of PacMan can be found in ‘Design’ section. The 3D model for the PacMan is created using MagicaVoxel [1]. Collider and Linecast are used to detect the interaction between PacMan and other GameObject. Several C# scripts are attached to the PacMan to handle user’s input and PacMan’s game logic. Complete codes for PacMan can be found in appendices.



*Figure 6.2.1 Making of PacMan in MagicaVoxel*

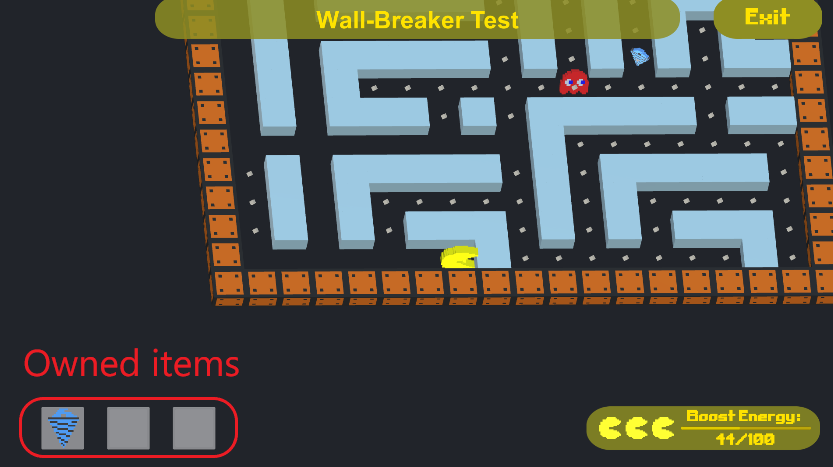
Figure 6.2.1 shows the making of PacMan’s 3D model in MagicaVoxel. MagicaVoxel is a modelling tool that provides modelling, rendering, painting and animation functions. All models used in the game are created using MagicaVoxel.



*Figure 6.2.2 PacMan in Unity game engine*

Figure 6.2.2 shows the PacMan in the Unity game engine. The red lines represent the Linecast in Unity. The Linecast is used to detect the existence of other colliders and it is used when PacMan tries to change its direction. For example, in figure 6.2.5, PacMan tries to turn left but Linecast finds that there is a wall (collider)on the left. Then, Linecast will return this result to the script and script will not allow PacMan to turn left.

The green lines represent the sphere collider of the GameObject. The collider is used to detect the interaction between PacMan and other GameObjects such as Ghosts and Items. When PacMan collides with another collider or trigger, the *OnColliderEnter(Collision)* function in PacMan’s script will be called [2]. Information about this collision and the GameObject is also passed to the Player script and PacMan will have corresponding actions. For example, when PacMan collides with a PacDot, the Player script will firstly get information about collision and PacMan knows that it collides with a PacDot. Then, PacMan will get one PacPoint and Boost Energy. Finally, the PacDot will be destroyed from the game scene.



*Figure 6.2.3 Owned items*

When player collides with an item object in the maze and player has an empty item box, PacMan will pick this item up and store it (except Energy Pellet) in the item box (figure 6.2.3). Player can use items after obtaining them. Each item has a unique function and details about items will be discussed in ‘Items’ section.

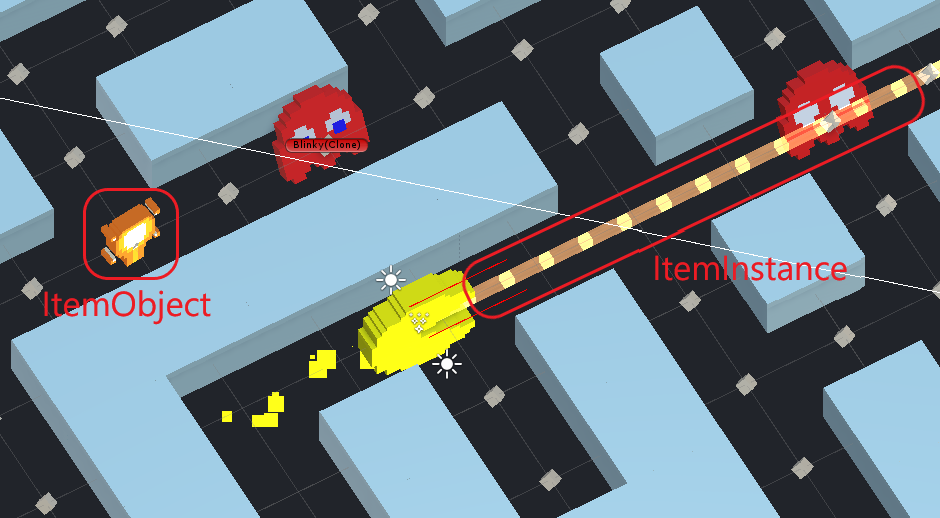


*Figure 6.2.4 Boost and boost energy*

Player can press ‘Space’ key and consume boost energy to make PacMan move faster. Figure 6.2.4 shows PacMan in boost mode and boost energy. PacMan can get boost energy by eating PacDot or energy pellet. Each PacDot provides 1 boost energy and each energy pellet provides 100 boost energy. In the boost mode, PacMan can move in 1.5 times of its original speed and each second in boost mode will consume 50 boost energy.

* 1. **Items**

During the implement stage, the design of items has been revised to meet requirements of the game better. In the design stage, a class: Item is designed to implement the functions of items. In the final product, Item class is removed and two class: ItemObject and ItemInstance are added to implement the item system. ItemObject describes GameObjects which have not been picked up by PacMan. ItemInstance describes GameObjects which are generated after using corresponding items. Figure 6.3.1 shows ItemObject and ItemInstance for Laser.



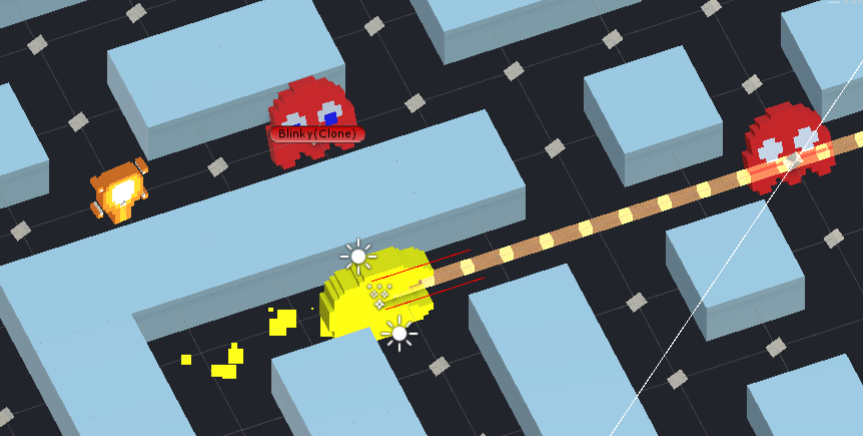
*Figure 6.3.1 ItemObject and ItemInstance*

The following table shows the structures for ItemObject and ItemInstance class. There are also 5 subclass of ItemObject: LaserObject, WallBreakerObject, PortalObject, PelletIObject and GrenadeObject. All of them inherit ItemObject and have their unique 3D model. Because EnergyPellet does not need an ItemInstance, there are 4 subclass of ItemInstance: LaserInstance, WallBreakerInstance, PortalInstance and GrenadeInstance.

|  |  |  |
| --- | --- | --- |
| **ItemObject** |  | **ItemInstance** |
| Name: String  Model: 3DModel  position: Vector3  trigger: Collider3D  unlocked: Bool | Name: String  Model: 3DModel  Position: Vector3  Collider: Collider3D |
| setPosition(): void  OnTriggerEnter(Collider): void  GenerateItemInstance(): void | OnColliderEnter(Collision): void |

Firstly, ItemObjects are placed in the maze. Then, if PacMan collides with the ItemObject and there is an empty item box, PacMan will pick it up and store it in an item box. When player presses corresponding number key, *GenerateItemInstance()* will be called to generate the ItemInstance. Finally, this ItemObject will be deleted from item box and ItemInstance in the game scene starts to interact with other GameObject.

Functions of items are not changed. ItemInstance is now responsible for functions of items and pseudo codes for each item has been discussed in ‘Design-Items’ section. The implement of these items follows their pseudo codes. Therefore, only ItemObject, ItemInstance and key methods used in the items will be discussed in this section.



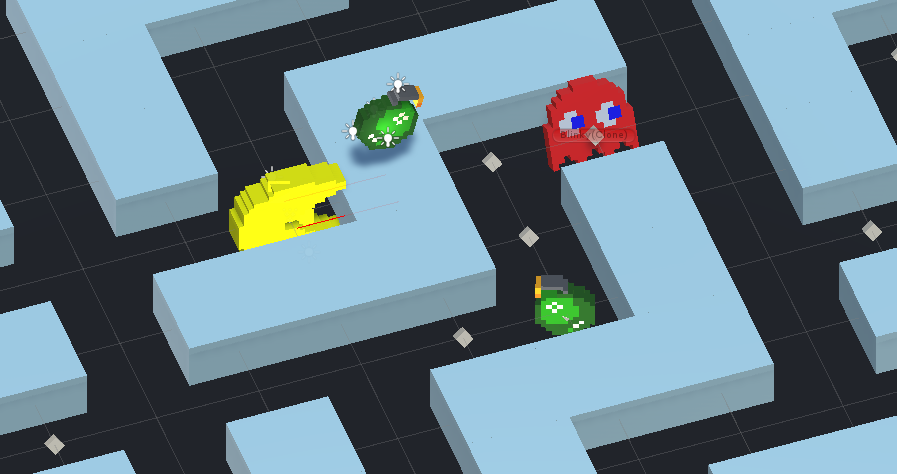
*Figure 6.3.2 Laser*

Figure 6.3.2 shows Laser item in the game, including LaserObject and LaserInstance. When *LaserObject.GenerateItemInstance()* is called, LaserInstance object will be generated in front of the PacMan and move forward in a high speed. When LaserInstance collides with any other collider, *OnTriggerEnter(Collider)* and *OnColliderEnter(Collision)* will be called. If LaserInstance collides with a wall, the part of laser that collides with wall will be destroyed. If LaserInstance collides with a Ghost, the Ghost will be disabled for 15 seconds. If LaserInstance collides with other GameObjects, nothing will happen. This item gives PacMan an ability to disable many ghosts with one item. However, ghosts are not destroyed so PacMan leave these ghosts before they awake.



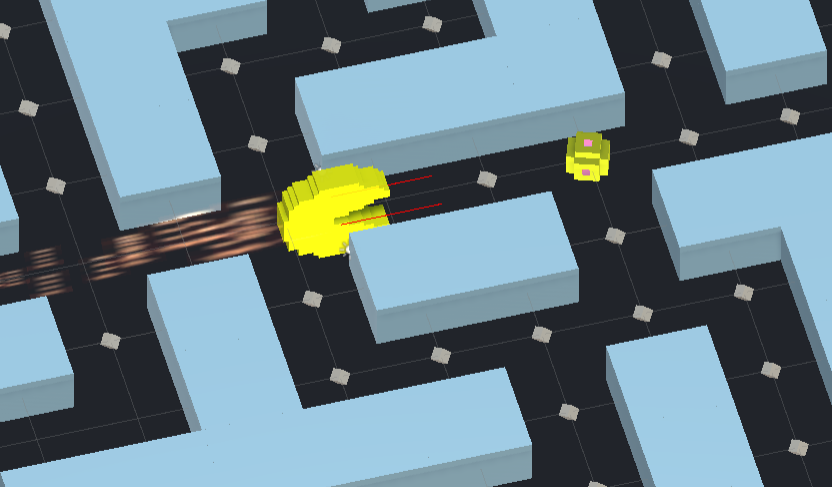
*Figure 6.3.3 Wall-Breaker*

Wall-Breaker item, including WallBreakerObject and WallBreakerInstance are shown in figure 6.3.3. When *WallBreakerObject.GenerateItemInstance()* is called, WallBreakerInstance object will be generated in front of the PacMan and move with PacMan. When WallBreakerInstance collides with any other collider, *OnColliderEnter(Collision)* will be called. If WallBreakerInstance collides with a wall, the wall and the WallBreakerInstance will be destroyed. If WallBreakerInstance collides with a Ghost, the Ghost and the WallBreakerInstance will be disabled for 15 seconds. If LaserInstance collides with other GameObjects, nothing will happen. Wall-Breaker is a powerful weapon and it has two different usages: destroy wall or ghost. Player should consider carefully before using it.



*Figure 6.3.4 Grenade*

Figure 6.3.4 shows GrenadeObject and GrenadeInstance in the game scene. When a Grenade is used, a GrenadeInstance object will be generated in the position of PacMan. The GrenadeInstance will fly over the wall in front of PacMan and then move forward. When the GrenadeInstance is moving, if it collides with a Ghost, both Grenade and the Ghost will be destroyed. If GrenadeInstance collides with a wall, the GrenadeInstance will be destroyed and a new GrenadeObject will be created in the empty cell of the collision. PacMan needs to move to the collision point to get grenade back. This item can help PacMan to destroy ghosts from a safe position but it may be difficult to aim the target.



*Figure 6.3.5 Energy Pellet*

Figure 6.3.5 shows the ItemObject of energy pellet. Energy pellet has no ItemInstance because energy pellet will provide PacMan 100 boost energy and it will always be used when PacMan collides with it. In addition, the energy provided by the pellet will ignore the energy capacity of PacMan so player can make use of the pellet without considering PacMan’s current energy.

1. **Ghosts**
2. **User Interface**
3. **Game Loop**

[1] MagicaVoxel. (2018, 1 Nov). *MagicaVoxel*. Available: <https://ephtracy.github.io/index.html?page=mv_main#>

[2] Unity. (2018, 2 Nov). *Collider.OnCollisionEnter(Collision)*. Available: <https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html>