

# Cabbage World

Jordan Ansari

## Contents

1. GameObjects .....	3
Cabbage Spawner.....	3
Cabbage.....	3
Dwarf Spawner .....	3
Dwarf .....	3
Game Manager.....	4
Badger .....	4
Snake.....	5
2. UML Diagrams.....	7
3. Key Features .....	10
4. New Entity: Snake .....	11
5. Fancy Points .....	11

# 1. GameObjects

## Cabbage Spawner

Spawns the cabbages at the beginning of the game and deals with their deletion.

### *Transform*

Only there because it has to be. The spawner is placed at 0, 0, 0 for simplicity's sake.

### *Cabbage Manager (Script)*

The script that spawns cabbages. By default, it spawns 6 in a random, semi-even spread. It also returns the nearest cabbage if requested, handles the eating of a cabbage, and stores the amount of cabbages left/eaten.

## Cabbage

The main goal of the dwarves, and what the player is trying to protect. They just sit there.

### *Transform*

Stores the position of the cabbage.

### *Sprite Renderer*

Draws the green circle that represents the cabbage.

### *Circle Collider 2D*

Exists so other game objects can know if they're colliding with the cabbage. The cabbage is on a separate layer so the player can move over them.

### *Sprites-Default (Material)*

There by default.

## Dwarf Spawner

Spawns the dwarves randomly around the map and manages their escaping.

### *Transform*

Only there because it has to be. The spawner is placed at 0, 0, 0 for simplicity's sake.

### *Dwarf Manager (Script)*

The script that spawns dwarves, the main enemy. It places dwarves every few seconds using a coroutine at a random location outside of the camera's view. Also handles the escaping of the dwarves if it's time for them to run away.

## Dwarf

The main enemy— brown squares that are after the badger's cabbages. They try to eat the cabbages and nap for a second once they do, but if the badger eats them, it gets a speed boost.

### *Transform*

Stores the location of the dwarf. They spawn at random locations and move towards the cabbages, so this keeps track of where they are.

#### *Sprite Renderer*

Draws the brown square that represents the dwarf.

#### *Box Collider 2D*

Used for collision, so it knows when it's colliding with other objects.

#### *Rigidbody 2D*

Used for the movement of the dwarf, by setting the velocity based on where the nearest cabbage is.

#### *Dwarf Movement (Script)*

Handles the movement of the dwarf with a default speed of 2.5f. It looks for where the nearest cabbage is and moves towards it, but if the player is too close, it adds the vector in the opposite direction to its movement, so it essentially tries to take a berth around the player.

#### *Dwarf Hits Thing (Script)*

Handles if a dwarf collides with something. If it hits the player, it gets eaten, but if it hits a cabbage, it eats the cabbage.

#### *Sprites-Default (Material)*

There by default.

#### *Game Manager*

Handles the spawning of the player, the tracking of progress, and the end of the game.

#### *Transform*

Only there because it has to be. The manager is placed at 0, 0, 0 for simplicity's sake.

#### *Game Manager (Script)*

Handles the starting and ending of the game by spawning in the player and keeping track of how many dwarves were eaten. If 5 dwarves have been eaten, it spawns the snake. If all the cabbages are gone, it freezes the player, makes everything run away, and displays the amount of dwarves eaten.

#### *Badger*

The player character, represented by a white circle. Can be moved with WASD/Arrow keys, and can eat dwarves. Trying to defend its lovely green cabbages from the evil dwarves.

#### *Transform*

Stores the location of the badger. The player is in control of the movement, so this keeps track of where they are.

#### *Sprite Renderer*

Draws the white circle that represents the badger.

#### *Circle Collider 2D*

Used for collision, so it knows when it's colliding with other objects.

### *Rigidbody 2D*

Used for the movement of the badger, by setting the velocity based on the player's input. Also allows it to push the snake around.

### *Badger Movement (Script)*

Handles the movement of the player by taking in the keyboard input and adjusting the Rigidbody's velocity accordingly. The default speed is 4.5f, but this gets sped up when the player eats a dwarf and slowed down when it gets bit by the snake or a dwarf eats a cabbage.

### *Sprites-Default (Material)*

There by default.

### *Snake Spawner*

Spawns and manages the snakes, including telling them to run away.

### *Transform*

Only there because it has to be. The spawner is placed at 0, 0, 0 for simplicity's sake.

### *Snake Manager (Script)*

The script that spawns snakes. It places snakes whenever the game manager tells it to (every time 5 dwarves have been eaten). It also handles their running away by telling every instance of the snakes to move forward.

### *Snake*

Vicious snake that doesn't really care about cabbages, but think the badger looks quite tasty. He'll settle for dwarf if he has to, though. Searches the farm aimlessly until he spots his prey.

### *Transform*

Stores the location of the snake. The snake rotates until it spots the player, so this keeps track of where it is

### *Sprite Renderer*

Draws the impeccable artistic masterpiece that represents the snake.

### *Capsule Collider 2D*

Used for collision, so it knows when it's colliding with other objects.

### *Rigidbody 2D*

Used for the movement of the snake, by setting the velocity and angular velocity depending on if the snake sees the player.

### *Snake Movement (Script)*

Handles the movement of the snake. It's spawned outside the camera and rotates slowly by setting the angular velocity until the player is within line of sight. Then, it rushes forward for 0.5 seconds before either continuing to rotate if the player is out of sight or rushing forward if it still sees the player.

*Sprites-Default (Material)*

There by default.

*Eye*

The eye of the snake.

*Transform*

Stores the location of the snake's eye.

*Sprite Renderer*

Draws the black circle that is the snake's eye.

*Sprites-Default (Material)*

There by default.

*Tongue*

The main part of the snake's tongue.

*Transform*

Stores the location of the snake's tongue.

*Sprite Renderer*

Draws the red rectangle that is the snake's tongue.

*Sprites-Default (Material)*

There by default.

*Fork 1*

The top section of the forked part of the snake's tongue.

*Transform*

Stores the location of the top part of the snake's tongue's fork.

*Sprite Renderer*

Draws the skewed red rectangle that is top part of the snake's tongue's fork.

*Sprites-Default (Material)*

There by default.

*Fork 2*

The bottom section of the forked part of the snake's tongue.

*Transform*

Stores the location of the bottom part of the snake's tongue's fork.

*Sprite Renderer*

Draws the skewed red rectangle that is bottom part of the snake's tongue's fork.

Sprites-Default (Material)

There by default.

*Front Hitbox*

Used for determining if the snake has hit something.

Transform

Stores the location of the front of the snake, which is used for determining if it bit someone. It's attached to one end of the snake.

Snake Hits Thing

Detects if the front of the snake hit something. If it hits the badger, the badger gets slowed down (as if a dwarf had eaten a cabbage), and if it it hits a dwarf, the dwarf gets eaten (but the player does not speed up).

Capsule Collider 2D

Used for collision to detect if the front of the snake has hit anything.

Sprite Renderer

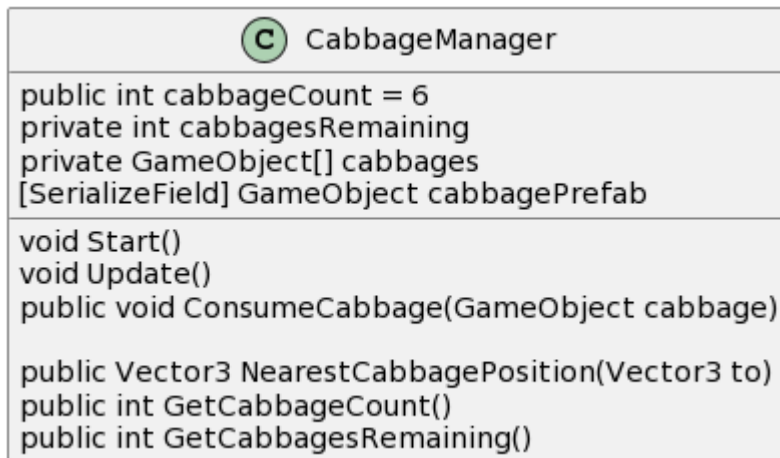
Draws an invisible circle.

Sprites-Default (Material)


There by default.

## 2. UML Diagrams


CabbageManager




## DwarfManager

 DwarfManager
<pre>private bool dwarfSpawn = true [SerializeField] GameObject dwarfPrefab</pre>
<pre>void Start() void Update() private IEnumerator SpawnCoroutine() public void DwarvesSuccess()</pre>

## DwarfMovement

 DwarfMovement
<pre>public float speed = 2.5f  private Rigidbody2D body private BoxCollider2D square  private GameManager gm  private bool napping private bool runningAway = false private Vector3 randomRunAway</pre>
<pre>void Start() void Update() public void RunAway() public void BeginNap() private IEnumerator Nap()</pre>

## DwarfHitsThing

 DwarfHitsThing
<pre>private void OnCollisionEnter2D(Collision2D collision)</pre>



## GameManager

C GameManager
<pre>[SerializeField] GameObject badgerPrefab GameObject badger private CabbageManager cm private DwarfManager dm private SnakeManager sm  private int eatenDwarves = 0</pre>
<pre>void Start() void Update() public GameObject GetBadger() public Vector3 NearestCabbagePosition(Vector3 currentPos) public void BadgerEatsDwarf(GameObject dwarf) public void DwarfEatsCabbage(GameObject dwarf, GameObject cabbage)</pre>


## BadgerMovement

C BadgerMovement
<pre>public float speed = 4.5f private Rigidbody2D bean private CircleCollider2D circle</pre>
<pre>void Start() void Update() public void IncreaseSpeed() public void DecreaseSpeed()</pre>


## SnakeManager

C SnakeManager
<pre>[SerializeField] GameObject snakePrefab</pre>
<pre>public void DwarvesSuccess() public void SpawnSnake()</pre>

## SnakeMovement

 SnakeMovement
<pre>private Rigidbody2D body private CapsuleCollider2D capsule private bool seesBadger = false private bool runningAway = false private int rotationDirection = 1</pre>
<pre>void Start() void Update() private IEnumerator ShootForward() public void RunAway()</pre>

## SnakeHitsThing

 SnakeHitsThing
<pre>private void OnCollisionEnter2D(Collision2D collision)</pre>

## 3. Key Features

I designed this game to have a fairly recognizable pattern in the code. First, every prefab has its own manager (Badger, Cabbage, Dwarf, Snake). These managers are generally responsible for making the game run—the Badger’s manager is GameManager, and it spawns in the badger, counts how many dwarves have been eaten, and handles the ending of the game. Additionally, each object that moves has a Move script, though the way these work depends on the object. Finally, both enemies have “(Enemy) Hits Thing” scripts, which deal with what the enemies should do if they collide with something. This is both useful and a problem with the design, since the snake eats the dwarf, and while I put that behavior in SnakeHitsThing.cs, it could have gone in either, and putting the behavior in both would lead to a race condition for destroying the dwarf.

I used Coroutines for the spawning of dwarves, the napping of dwarves, and the moving of snakes. The dwarf spawning has a boolean that gets activated once the old spawn’s coroutine has finished. The dwarves’ movement uses a coroutine that lasts for 1 second and halts its velocity. Finally, the snake uses a coroutine for its movement to jump forward for 0.5 seconds before performing its check again.

The only instance of a raycast is on the Snake, which has a functionally limitless ray coming out of its front. This pierces through multiple objects, and if one of the objects it hits is the player, then the snake shoots forward.

I’m overall satisfied with my design, but I added a GameManager too late and had to port over a lot of the functionality from the other managers to it for simplicity’s sake. Were I to do this again, I would add the GameManager earlier and have it do more, possibly even collision

handling so there's never a scenario like the snake eating the dwarf where it's unclear what should happen.

#### 4. New Entity: Snake

A snake spawns outside of the camera's view for every 5 dwarves that the player has eaten. When spawned, it rotates slowly in a random direction. It has a raycast of length 100 coming out from its front end, and whenever this intersects with the player, the snake's velocity is set to 13 in the direction it's facing for 0.5 seconds. After this interval, it repeats this behavior (rotates in place if it doesn't see the player, shoots forward if it does).

If the front of the snake intersects with the player, their speed is reduced by 10%. If the front of the snake intersects with a dwarf, the dwarf is eaten. The snake can collide with both the dwarves and the player, so it can be an obstacle for both of them while it rotates.

#### 5. Fancy Points

- Spawn dwarves off camera, even if the camera moves:
  - The camera doesn't move in this, but the dwarves always spawn off-camera by randomly picking one of the four sides of the rectangle and spawning within a random range (i.e.  $x=10$  to  $15$ ,  $y = -5$  to  $5$ ).
- Dynamically spawn cabbages in the beginning at reasonable places
  - The CabbageManager spawns each cabbage at a different angle and distance from itself. It increments the angle after each spawn and randomly selects a distance. The new angle guarantees the cabbage will spawn at a different location than the one before it. Finally, it multiplies the  $x$  value by 3 to spread them more across the map.