Third Workshop Demo Information

By Peter Mitchell

The content discussed in this document can be found at:

Unity Demo Project Github: https://github.com/Squirrelbear/ThirdWorkshopDemoProject

Chad's Challenge Github: https://github.com/Squirrelbear/ChadsChallengeDemoCopy

Contents

1 Control Summary for Scenes

Each of these sections summarise the difference scenes you can find in the Scenes folder along with a summary of the controls you can use for each.

1.1 TDBaseMap Scene

The TDBaseMap scene provides a simple scenario with a tower defence scene played out on a simple terrain where enemies come out from a cave and travel a defined path to the end. You can move the camera around, and spawn towers to prevent the enemy waves. The enemy waves are predefined and come in a set sequence.

Controls:

- Move camera = Arrow keys.
- Rotate camera = Hold right mouse button and move.
- Reset Camera = 0
- Spawn a Basic Tower = 1
- Spawn a Frost Tower = 2
- Spawn a Swarm Tower = 3
- Spawn an Explosive Tower = 4
- Play Explosion of Rocks Sequence = J

1.2 BreakdownDemo Scene

The Breakdown Demo scene provides examples of improved code for the tower defence. The components are designed to have individual roles and uses events to handle state changes. You play as a spider moving through the scene.

Controls:

- Move character = WASD
- Jump = Space
- Toggle camera between 1st/3rd person = P
- Toggle show tower range = Click on the towers
- Spawn tower = Jump onto the tower placement and click on the preferred tower.

1.3 EventDemo Scene

The EventDemo scene provides a couple of examples of types of events using both Unity events and those provided natively via C#. You play with the same character controller used in the BreakdownDemo. There are no special controls, you can walk into the cubes in front of the doors to open them, click on the buttons on screen to either reveal all the hidden collectibles or open all doors, or once collectibles have been revealed you can walk through them to collect them all.

Controls:

- Move character = WASD
- Jump = Space

1.4 DungeonDemo Scene

The dungeon demo is in some ways similar to the 3D Platformer procedural demo. It generates a random layout of rooms connected via corridors. Then places items that do nothing randomly over the map. The player is placed on the map and the player can move between cells on the grid.

Controls:

- Move character = WASD
- Generate map = G
- Re-randomise objects = O
    - Note that you can make objects spawn in corridors too if you select the DungeonManager object and near the bottom of Dungeon Generator click Make Corridors Rooms. Then press G at least once to make it repopulate.

2 GameObject Composition

2.1 Single Responsibility Principle

There are two version of the Tower Defence game content. One is the TDBaseMap Scene paired with the content in the TowerDefence folder and the other is the BreakdownDemo Scene paired with the content in the BreakdownDemo folder.

The Single Responsibility Principle is a principle that promotes every module/class/function in programs have the responsibility of a single part of the program's function and should encapsulate that part. When you are doing this in Unity, you can make your scripts have single jobs. The TDBaseMap does not really follow this so much. The classes are bloated with combining multiple elements into single classes. For example, the AIWayfinder is actually handling the health, movement, debuff management, and on death actions for the AI. You will find these have been broken up into distinctly separate classes in the BreakdownDemo as a better example.

When possible, it will make it easier to make changes if you have your scripts perform one distinct job. You will under time pressure probably not be able to do this consistently. You can always refactor later if you want. It is more important you get things to a working state first than to strictly follow principles.

Note that this is part of a set of principles known as

2.2 General Tips for Managing Objects

2.2.1 Object Pooling

Object pooling is an approach to dealing with having a very large number of objects that you need to create/destroy frequently. This can be used with elements such as projectiles. The TowerBehaviour in the TowerDefence Script folder shows an example of this. You can instantiate enough objects at the first possible point and initially disable them all. Then when you need one of those objects you can setup its position and other properties and shift it to be on the active list. Then once the object is finished its job you can move it back to the inactive list. This can continue forever. Choosing the number of elements to instantiate can either be based on an expected maximum, or you could have the objects be instantiated if the inactive list is empty when it tries to spawn a new one. Then you will only spawn as many as you need and they will gradually spawn at first.

The relevant code from TowerBehaviour.cs for this tip are:

```
public List<GameObject> offProjectiles;
public List<GameObject> onProjectiles;

    void Awake()
{
    onProjectiles = new List<GameObject>();
    offProjectiles = new List<GameObject>();
```

```
        if (towerType != TowerType.Frost)
        {
            for (int i = 0; i < 10; i++)
            {
                offProjectiles.Add(spawnProjectile());
            }
        }
    }

       private void fireProjectile()
    {
        // Don't fire if the tower is completely destroyed.
        if(towerType == TowerType.Frost)
        {
            slowAllTargetsInRange(maxRange, towerDamage);
        }
        else if (currentTarget != null)
        {
            GameObject projectile = null;
            if(offProjectiles.Count == 0) {
                projectile = spawnProjectile();
            }
            else
            {
                projectile = offProjectiles[0];
                offProjectiles.RemoveAt(0);
            }
            onProjectiles.Add(projectile);
            ProjectileBehaviour projectileBehaviour =
projectile.GetComponent<ProjectileBehaviour>();
            projectileBehaviour.setTarget(currentTarget, firingPosition);
        }
    }


    public void resetProjectile(GameObject projectile)
    {
        onProjectiles.Remove(projectile);
        offProjectiles.Add(projectile);
    }
```

2.2.2 Sensible Hierarchy Object Spawning

If you just use Instantiate(), the objects are spawned in the top level of the hierarchy with no parent. You can set the parent of any object after it is Instantiated. For example, also in the TowerBehaviour.cs you can find this in the spawnProjectile() method. Setting the object's transform.parent to the transform of the object you want to place it under in the hierarchy makes the scene easier to follow. In this case all the projectiles for the single tower are controlled by that individual tower so they appear under that tower in the hierarchy.

```
    private GameObject spawnProjectile()
    {
        var projectile = Instantiate(projectilePrefab, transform.position,
transform.rotation);
        projectile.transform.parent = gameObject.transform;
```

```
        ProjectileBehaviour script = projectile.GetComponent<ProjectileBehaviour>();
        script.setTower(gameObject);
        script.destroyProjectile();
        return projectile;
    }
```

2.2.3 Object Reference Caching

If you need to access either components or objects that are from the same source repeatedly you should try to cache a reference. In particular if you are using methods that "Find" game objects constantly those can be very expensive in terms of resources. If appropriate you can create a variable in your class that is constantly using the other object/objects/components etc. And then you can use that reference instead of having to look up information constantly. You basically just have to create a variable of the type of either GameObject or class name for the matching type you are trying to store. Then store that reference either by using the inspector to set it in the scene, or by setting it in the Start()/Awake()/OnEnable() function/s whichever is most appropriate. Sometimes you may do this when performing some other action. For example, the projectiles when fired at a target cache a reference to the tower they were fired from and a reference to the object they are fired at.

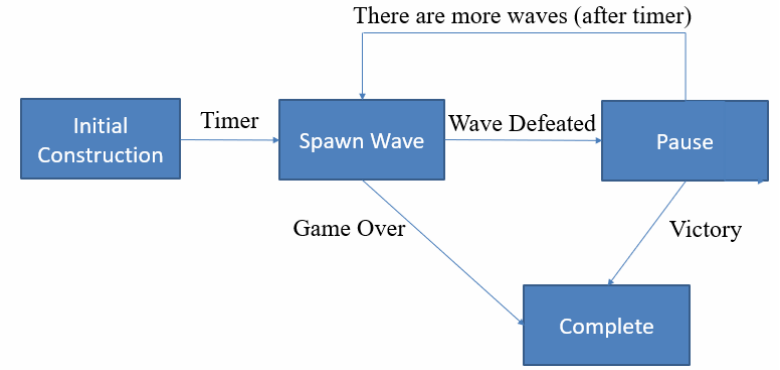
3 Tower Defence

3.1 Summary of Core Tower Defence Parts

Tower Defence games typically involve the following types of elements that have been incorporated into this example.

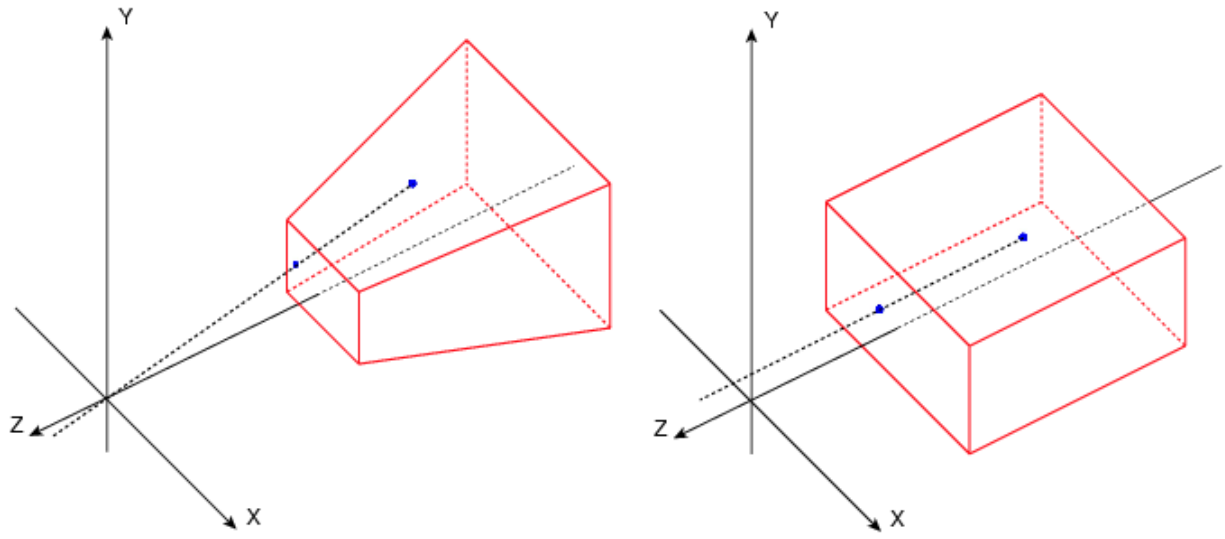
- Towers
    - Fire projectiles.
        - Projectiles can fire at single targets or affect multiple targets.
        - Projectiles can either damage the single target they hit, or damage all those in a range around the impact.
        - Projectiles can include ones that deal no damage but apply debuffs such as a slow.
        - Firing of projectiles has a delay between shots.
        - Targets are selected and then kept as the current target until the target is out of range, or no longer valid.
    - Can be placed at specific places on the map.
    - Can be picked up and moved.
- Enemies
    - Spawn from a spawner point.
    - Move through a sequence of waypoints with different speeds for each type of enemy.
    - Enemies have health that varies depending on the type of enemy.

    - Enemies are grouped into waves with delays between them to make spawning sensible.
- Game Flow
    - Initial Construction -> Spawn Wave -> Intermission -> Repeat Wave/s and Intermission -> Game End
    - This would look like the State Machine diagram shown below.



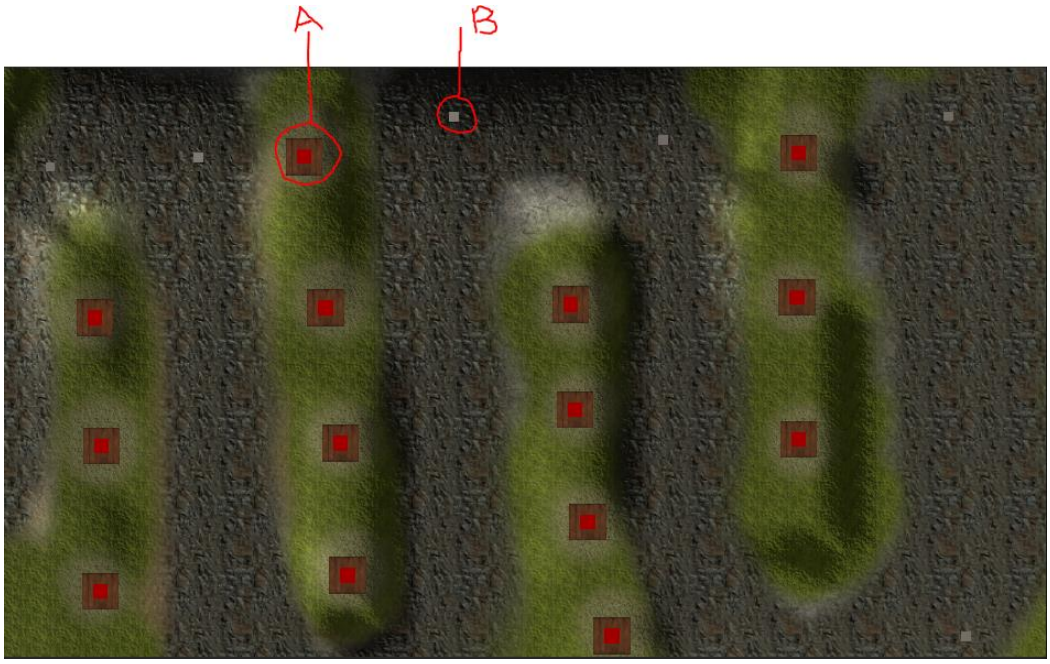
3.2 Notes on Camera Movement

- transform.position (x,y,z movement)
- transform.localEulerAngles (x,y,z rotation)
- Restricting bounds can be applied with Mathf.Clamp(v, min, max)
- Perspective Camera (left) vs Orthographic (right)

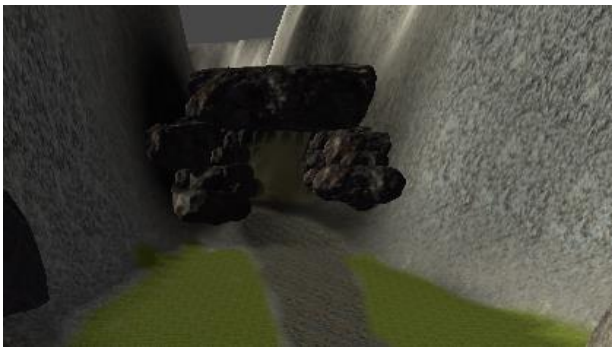


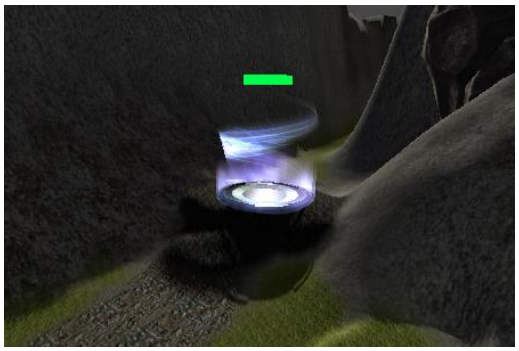
3.3 Summary of GameObjects

You can see the default view of the game when it starts below with A being an example of a TowerSnapPoint where towers can be placed. B shows an example of a Waypoint used by the AI to navigate from point to point until the end. These are set to be invisible after the game begins.

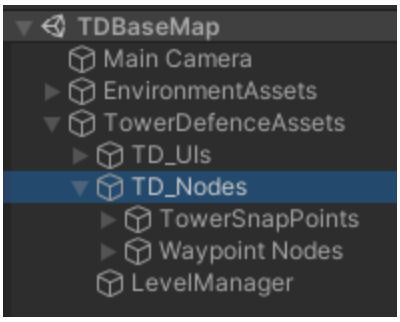


The start and end of the waypoints are shown below. A cave shrouded with a dust particle effect with rocks that also explode outward when either J is pressed or when the final boss spawns. They travel until the teleporter pad is reached where they reduce the green health bar with each one that reaches it.



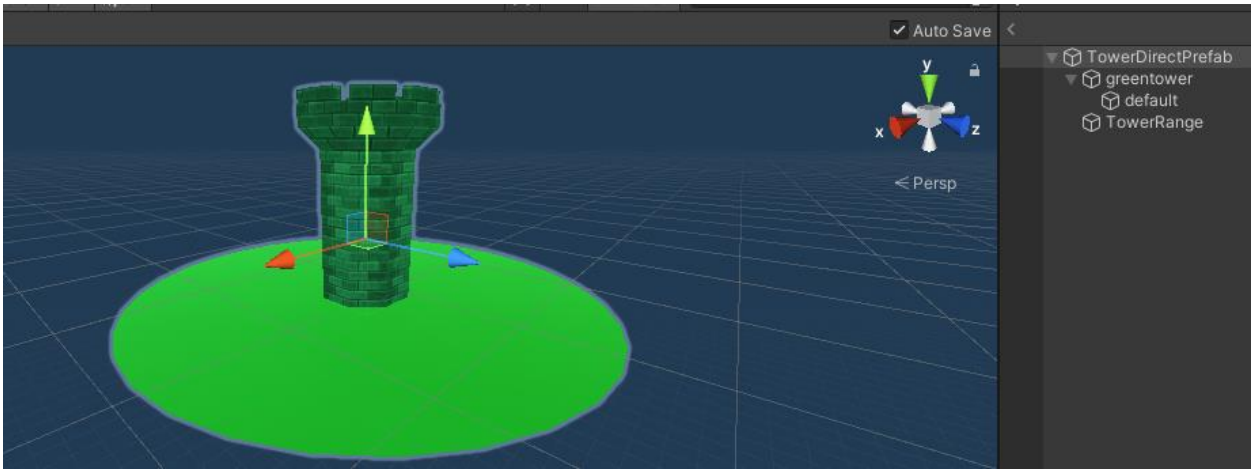


The scene as shown below has elements nested under appropriate headings with all snap points and waypoints under appropriate empty game objects. The AISpawner is also nested under the waypoints You will also notice the LevelManager that is responsible for coordinating the scene.



3.3.1 Tower Prefabs

There are four different towers the Basic tower (TowerDirectPrefab), Slow tower (TowerFrostPrefab), Swarm tower (TowerSwarmPrefab), and Explosive tower (ExplosiveTowerPrefab). Below is an example from the Basic tower showing the hierarchy used for the towers. Followed by a summary of the different parts that make up each tower.



- TowerDirectPrefab
    - Box collider with a cube mesh to allow for selection.
    - TowerBehaviour Script Properties are mostly set by the script when the object is created. The properties that can be changed in the inspector are:
        - Tower Type: This dictates what the other properties are set to.
        - Projectile Prefab: The object that is fired by the tower.
- greentower and default
    - The greentower object is an empty object with some scaling changed to modify the sizing.
    - Default contains the mesh and the texture.
- TowerRange
    - Contains a circular flat mesh that is expanded to represent the range the tower can hit. This is set via the TowerBehaviour.
    - TowerRangeBehaviour manages the opacity of the mesh.

- The one exception to this structure is the TowerFrostPrefab. Because the frost tower does not actually fire projectiles, it instead pulses a slow and that is visualised by a FrostTowerAttack object attached with the effect.

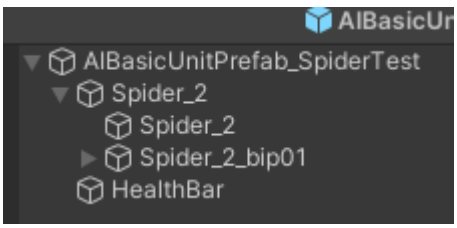
3.3.2 Projectile Prefabs

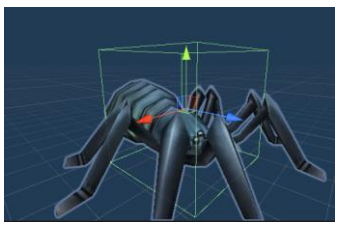
There are three different types of projectile prefabs. The one shot by the Basic tower (ProjectileDirectPrefab), one shot by the Swarm tower (ProjectileSwarmPrefab), and the one shot by the Explosive tower (ProjectileAoEPrefab). There are all simply single game objects with scaled down cubes coloured using different materials for each one. They have no collider enabled. The logic for the projectiles and their configuration is all handled via the ProjectileBehaviour script. The following properties can be changed for the projectiles.

- Path Type: Direct or Swarm. The Direct option moves continually forward changing its facing to the target. The Swarm option makes the turn speed slow to gradually reach the target causing the projectile to circle around the target a bit.
- Move Speed: The speed the projectile moves to the target.
- Rotation Speed: The turn speed of the projectile.
- Damage Type: Single Target or AoE Radius
- AoE Damage Radius: When the AoE Radius is selected this is the distance around the target that are also damaged by the projectile exploding.

3.3.3 Enemy Prefabs

The prefabs for all the enemies consist of three core elements. As seen below the AIBasicUnitPrefab_SpiderTest object representing the basic enemy has a parent object that contains a cube with the box collider enabled, but mesh renderer off. This box provides the collisions for movement with a Rigidbody that is controlled using an AIWayfinder script. The cube that doesn't match the same full size of the spider (even more so for the other enemies). This is because the spiders should be able to group up tightly. And to do that the collision material needs to be smaller than the model. Nested under the object with the cube is the model (in this case Spider_2, with an object for the model, and then a sequence of nodes that map to the joints if that is necessary to use). And importantly a HealthBar that faces the camera to show the health of the enemy visually.





The health bar works by having a simple plane that is scaled initially to represent full health. Then the scale is changed relative to the percent of health for the enemy. Therefore, as the enemy health reduces the plane shrinks and makes the health visually appear to be decreasing.

3.4 Summary of Scripts

One of the big issues with the way these scripts are set up is that they are all linked requiring all of them to be there. This is mostly via the update(deltaTime) methods. This is all removed as a point of significance from the BreakdownDemo. This is mostly an example of what you should try to avoid doing in Unity unless necessary, but it is worth showing a worse way to do it and then the better way.

- AISpawner: Takes information from the LevelManager and spawns waves of enemies that are initially disabled. Then gradually enables them based on timings sent along with the definition.
- AIWayfinder: This is the script responsible for moving enemies, managing their health, their state changes with movement speed, and their animation control.
- CameraBehaviour: This script manages both the movement of the camera itself, and the placement of towers. This script manages the snapping of towers that are being moved/placed so you can look at how that is done in here if you want.
- EndPointBehaviour: This is a script that is on the teleporter waypoint to manage the health. When AIWayfinders reach the object with this attached they deal damage by calling applyDamage().
- ExplosionSimulationManager: This script manages the collection of rocks that explode outward by using a collection of rocks and having positions that they will travel to once exploded out.
- HealthBarBehaviour: This simply forces the object to always face the camera and allows setting of health to force scaling onto the object based on health.
- LevelManager: This manages the overall game state with moving through the sequence of multiple waves and passes the information about those waves to the AISpawner. It also maintains management over all the towers that are active in the scene.
- ProjectileBehaviour: This manages the movement from whatever point the object starts at and moves toward the target. Once the target is reached it damages the enemy or enemies depending on the configuration of the script.
- RockMoveBehaviour: This script manages the individual movement of rocks. Once the rocks begin to move, they are scripted to arc toward the end position where they are supposed to end up.
- SpawnAndDestroyEffect: I am pretty sure this is unused in this version of the code, but this script can be used to spawn an object and then destroy an object after the timer. In the not striped down version of the code, this was for showing an effect when towers were placed.
- TowerBehaviour: This script defines the targeting behaviours including range, cooldowns, damage, and use of projectiles that make up how towers attack enemies.
- TowerRangeBehaviour: This script cycles the opacity to make the transparency amount change for the circle showing the range of a tower.

- TowerSnapBehaviour: This mostly just stores for a given TowerSnap point if there is currently a tower associated with that point and where the tower that is snaped to it should be placed.
- TowerStatusDialog: This manages the tooltip showing information about the selected tower. You will see it pop up when you select towers.
- WaveCommand: This script defines the data stored by each command. This is simply a pair of a time to wait (how long between spawns), and the ID of the unit to spawn that is mapped to a list of prefabs. This is used by the AISpawner and LevelManager.
- WayPointBehaviour: This acts as a simple linked list with each object pointing to the next object. The only functional code in this is that when it starts the object is set to be not visible.

4 Breakdown Demo (Improved TD)

The Breakdown Demo includes new versions of most scripts that can be considered improved over their counterparts. It does not include a couple of parts like the selection and moving of towers from the CameraBehaviour, or the damaging of the end point. There are also significant changes that add the 3D character controller. You can find all the added content related to this section under the BreakdownDemo folder.

4.1 3D Character Controller

You can find a Prefab containing all the setup for the 3D character in the BreakdownDemo/Prefabs folder called “PlayerPrefab”. This is based on the basic spider enemy with all the AI content removed and replaced with content to control how the player should be used. The Rigidbody is removed on the core object. The box collider is also not necessary but has just not been removed. The CharacterController doubles as both the collider and the component interacted with by scripts to move the object. The CharacterController is an object that is part of the core Unity content. It differs from elements like Rigidbody because it allows immediate stops.

The only default property changes in the CharacterController is the “MinMoveDistance” that is set to 0. This is necessary for the correct detection of the player being grounded in the DPlayerController script.

The DPlayerController script has the following properties you can change:

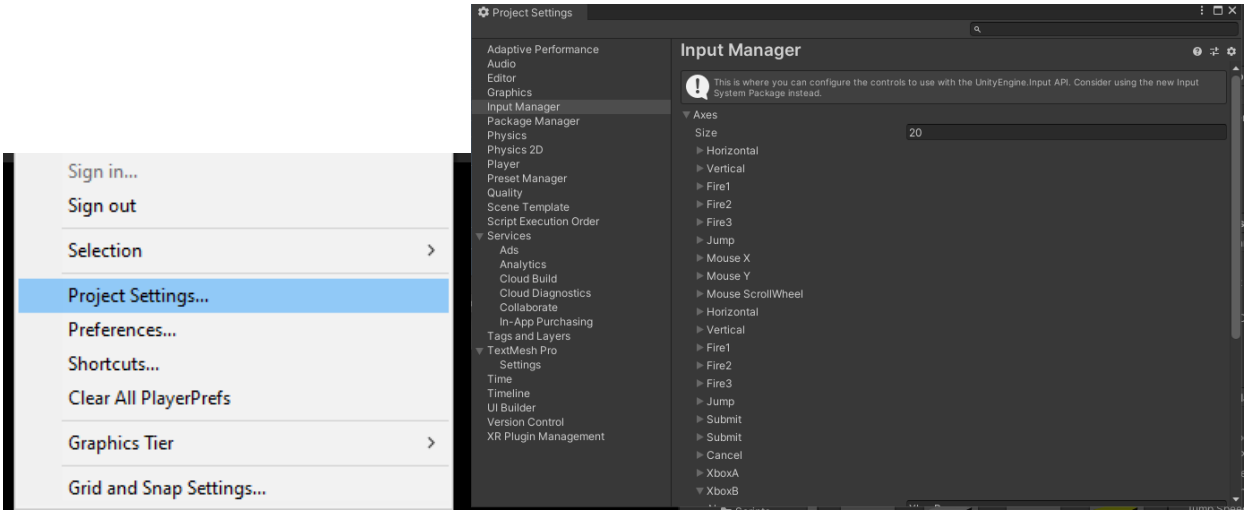
- PlayerSpeed: The speed the player moves forward in the current direction of facing when the W/D buttons are held or Up/Down arrows.
- Jump Height: The height the enemy jumps to.
- Jump Speed: The speed at which the enemy moves upward to the height. Note that if you change the speed from -3 it won’t actually go to the specified height.
- Gravity Value: You could change this if you want gravity to be more/less than normal.
- Turn Sensitivity: The speed at which the player can turn.

You can find a variety of method examples in this class that are both used and unused. I have included an example of both turning using keys (the default) and turning using the mouse (not used). There are also examples of how you can strafe (unused).

The DCameraSwap script references the ThirdPerson and FirstPerson cameras that are nested under the PlayerPrefab. It is important that only one of these objects is active. The purpose of this script is to toggle the active camera object through the list of cameras whenever P is pressed.

4.1.1 Finding the InputManager

You can change the controls used for different inputs and define more under the Project Settings. This can be found by going to Edit->Project Settings (as seen below on the left). This will show the dialog seen on the right. Where you can select “Input Manager”. The content on the right shows all the different inputs. The names listed there are what you refer to when saying things like Input.GetAxis(“NameHere”) or Input.GetButtonDown(“NameHere”).



4.2 Summary of Scripts

All these scripts can be found in the BreakdownDemo/Scripts folder. Most of the scripts have a D before the script name to make sure they are distinct from any other scripts in the project.

- DAISpawner: This is very similar to the original AISpawner script, but it does not require any association with the LevelManager script. It independently spawns all the objects as defined by the arrays near the top of the file. This is a partial merge from the LevelManager.
- DCameraSwap: Used by the Player controller to swap between first and third person cameras. This would work for any number of cameras.
- DDeathAnimator: This listens for the UnitDied event in DHealthBehaviour to begin animating the death. It is used for all the enemy prefabs. It allows definition of the time it takes to die (deathtime), the amount of drag causing rotation via the RigidBody while

dying (dragOnDeath), and a reference to the animation for the object to be played on death (deathAnimation).

- DDebuffBase: This is an abstract class meaning you can't directly create instances of it. This is used as a parent to inherit from for the DDebuffSlow and provides a simple definition that can be reused for any number of possible debuffs. The DDebuffManagerBehaviour class manages a collection of these. The DDebuffBase manages a name for the debuff, and a duration. applyEffect() is called when the debuff is initially applied. Then removeEffect() is called when it is removed. These two methods can be overridden to define the implementation of the debuff.
- DDebuffManagerBehaviour: Manages a list of DDebuffBase objects that are updated every Update() with a method to add debuffs. Those debuffs are not applied if they already exist.
- DDebuffSlow: The class extends from DDebuffBase allowing definition of a duration/speed multiplier for a debuff called "SlowDebuff". The applyEffect multiplies the speed by the specified multiplier and to undo it when the effect is removed it multiplies the speed by 1/multiplier.
- DDebuffHealthBarBeavhiour: The script for managing the health bar. It receives events from when the HealthBehaviour for the object triggers a UnitDied event (causing the health bar to hide), and the UnitHealthChanged causing an update to modify the health values.
- DHealthBehaviour: This class keeps track of a current and max health. It has two events a UnitHealthChanged that passes the (current, max, changeAmount) to anything that listens for the event. And when the unit dies it triggers a UnitDied event. The class has methods to provide an isDead() status, and changeHealthBy(amount). The later method does nothing if the amount would not cause a change in health due to max health or if it is already dead. The events are triggered as necessary with calculated actual amounts of change.
- DMoveToTargetBehaviour: Follows a list of waypoints by moving forward and turning toward the point. It listens for the DHealthBehaviour triggering a UnitDied event to stop moving. When the object finishes moving toward the object (meaning the currentWaypoint reaches null) it triggers TargetReached as an event. This is currently only used by the DAISpawner, but it could be used to do a variety of other events.
- DPlayerController: Already summarised in the 3D Character Controller section.
- DProjectileBehaviour: Provides options for defining a ProjectilePathType, a ProjectileDamageType, and a DebuffType. It is responsible for both the movement and events related to the collision. This could be split out into the incomplete DProjectileMoveToTarget.
- DTowerBehaviour: Similar to the TowerBehaviour script it does also trigger OnSelectedTower and OnDeselectedTower events.

- DTowerRangeAnimationBehaviour: Includes listening to events from the DTowerBehaviour to show/hide the visual effect.
- DTowerSnapPointBehaviour: Includes triggers for the TowerChoiceManager.
- TowerBuildSelectionBehaviour: Allows definition of the towerID to be spawned when a choice is selected. This is discussed further in the TowerChoice for 3D Character Controller section.
- TowerChoiceManager: Provides a manager to a collection of towers that are hidden/revealed based on interaction with the TowerSnapBehaviours using the character controller.

4.3 Changes to Towers/Projectiles

There are new prefabs for the towers, projectiles and tower snap points in the BreakdownDemo/Prefabs folder. A big difference is that now the frost tower does not have the effect tied to the prefab as a nested object. It is now a separate prefab. The towers no longer have any information about the damage they deal. That is now handled by the projectile attached to them. The DTowerBehaviour script allows for definition of the cooldown between firing of projectiles, the range the targets can be chosen at, the position they are fired from, a reference to the prefab that is fired at the target. An IsShooting property that makes the tower only shoot while it is true. And a RequireTargetToShoot property that is true for all towers except frost. The frost tower is special because its projectiles are spawned at the fire position and then destroyed after the effect ends.

The tower/projectile prefab pairs include:

- D_TowerDirectPrefab: Fires D_ProjectileDirectPrefab.
- D_TowerFrostPrefab: Fires D_FrostTowerAttackPrefab.
- D_TowerSwarmPrefab: Fires D_ProjectileSwarmPrefab.
- D_TowerExplosivePrefab: Fires D_ProjectileAoEPrefab.

4.4 Changes to Enemies

Enemies are more significant in terms of their script changes. The original enemies all used just a single script called AIWayfinder. In this modified set of scripts, they all have a DHealthBehaviour for managing the health, a DDebufManager to control any debuffs (really only the slow debuff), a DMoveToTargetBehaviour for moving between the sequence of waypoints, and a DDeathAnimator to handle the sequence of events that happen when the unit dies. The health bar also is using a modified script with the DHealthBarBehaviour that listens independently for updates to the health without requiring the DHealthBehaviour to be aware of it.

The modified prefabs included using these are:

- D_AIBasicUnitPrefab
- D_AIFastUnitPrefab

- D_AIDangerousUnitPrefab
- D_AIBossUnitPrefab
- D_AIMegaBossUnitPrefab

4.5 Tower Choice for 3D Character Controller

When playing in first person it doesn't make as much sense to be placing the towers using a raycast from the mouse. So instead, the alternate version of the choosing is done by walking into the snap point and the options appear to be clicked. This can be found either in the scene or as a prefab called TowerChoiceOptions. When walking into the red part of the tower snap point the options appear as seen. Then clicking on any of them will spawn that tower at the point where you currently are.

This is managed through a combination of the objects in the TowerChoiceOptions group, the script to call the TowerChoiceManager script via a singleton reference in DTowerSnapPointBehaviour, the TowerChoiceManager for showing/hiding the group and spawning the tower. And the TowerBuildSelectionBehaviour for handling the clicking on an option.

5 Events Example

The EventDemo scene shows a couple of specific examples of Events. These are based on the following two videos if you are interested in reviewing the original material. You can find all the scripts related to the demo under the EventExample folder.

Item Collection:

https://game.courses/c-events-vs-unityevents/?ref=15&utm_source=youtube&utm_medium=description&utm_campaign=unityevents

Doors: https://www.youtube.com/watch?v=70PcP_uPuUc

5.1 Summary of Events

Events take two forms. You can either use C# events or Unity events. The one you use depends on whether you want to modify anything inside the inspector. If you aren't wanting to use the inspector to modify what the events do you can just use the C# version.

C# Events:

- Define the event

  ```
  public event Action OnSomeEvent;
  ```

- Register methods to the event

  ```
  public void exampleMethod()
  {

  }
  ```

  Then in a method like OnEnable() or wherever you want to register it:

  ```
  OnSomeEvent += exampleMethod;
  ```

- Deregister methods to the event

  It is important to deregister if you are going to destroy an object that is linked to some other object that stays with the event. You can do this in a method like OnDisable().

  ```
  OnSomeEvent -= exampleMethod;
  ```

- Trigger the event

  Call the following when you want all the methods linked to the event to be triggered.

  ```
  OnSomeEvent?.Invoke();
  ```

Note that the ? operator here makes the statement the equivalent of writing:

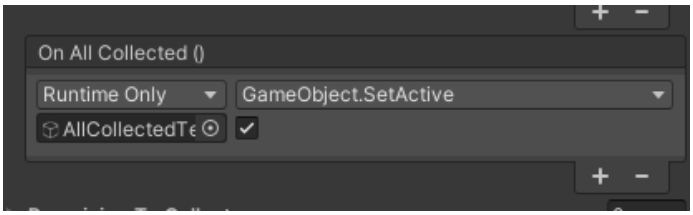
```
if(OnSomeEvent != null) {
    OnSomeEvent.Invoke();
}
```

This prevents needing to do a null check.

Unity Events:

- Define the event

  public UnityEvent OnSomeOtherEvent;
- In the Inspector you can set what happens during the event.



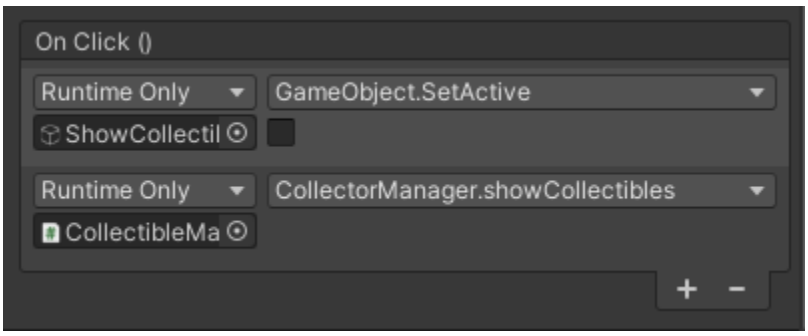
  Add/remove events with the +/- buttons. Most of the time you will set the dropdown to RuntimeOnly. Then drag in some object to be modified. Once you have an object or script linked to the event action the drop down will let you change the properties of many different parts of that object including calling methods. You can add as many different changes as necessary.
- Then when you want the event to execute you can just call:

  OnSomeOtherEvent.Invoke();

5.2 Collectibles Event Example

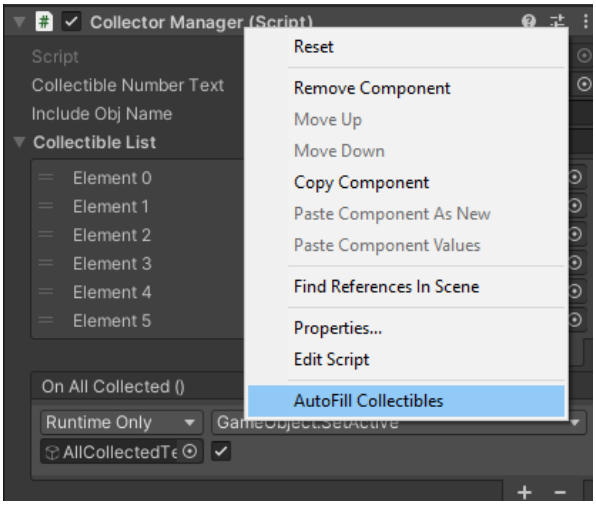
The Collectibles example demonstrates using events triggered when objects are picked up, revealing via a button UI action event, and an event triggered when all objects have been collected.

Firstly, the button that says “Show Objs” called ShowCollectiblesButton has an OnClick event as seen below. It hides the button by setting the object to not be active. Then it calls the showCollectibles() method in the CollectibleManager. This reveals all the collectibles ready to be collected.



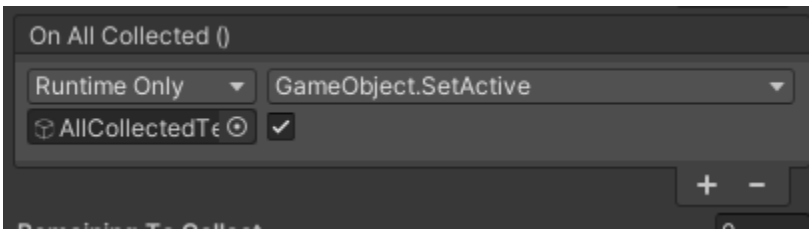
The CollectorManager sets up all the Collectibles as a list of all the objects and a list of those remaining to be collected. In the OnEnable() method the remaining list is set up and the OnPickup event in all the collectibles are registered to call the HandlePickup method in the CollectorManager.

Note that you can auto populate the list of collectibles via the method AutoFillCollectibles. You can see this in action via a Context Menu visible when you right click on the component. You can add your own features like this to the Unity UI if you feel like it is useful.



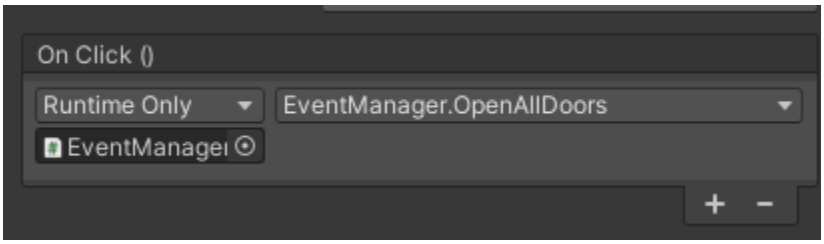
In the Collectible script the most important part is the OnTriggerEnter method. This triggers the OnPickup event calling the HandlePickup in CollectorManager, and then sets the object to be inactive.

The HandlePickup method removes the object that was passed. Notice how in this case a parameter is passed because the definition of the event in Collectible was Action<Collectible> this means that any method attached as a listener to the event must have a single parameter of type Collectible. The HandlePickup method updates the text to show the new number of remaining elements to pick up and then if there are none remaining the OnAllCollected event is triggered. This is a UnityEvent that is defined in the inspector as shown below. It just enables the AllCollectedText to show the green success message. You could have it do other things if you wanted very easily though.



5.3 Doors Event Example

The doors example is mostly important because it shows how you can share an event and then use a parameter to determine which objects react to the event. There is a button in the UI that has an OnClick as seen below that calls the OpenAllDoors method in the EventManager class. This allows all the doors to open regardless of state.



The EventManager has two statically defined events meaning they can be accessed from anywhere. These are Invoked via the associated static methods meaning the EventManager is simply there to maintain a collection of associated events. The OnDoorTriggerAllEvent will send a message to all the doors that are registered and the OnDoorTriggerEvent sends the doorID that will only open the door with that doorID.

The Door class listens to both the events with separate methods to handle each event. These are OpenDoor() and OpenDoorByID(). They are registered in the Start() method to the events. The doorID is set in the inspector. Note that the OnDisable() method removes the events.

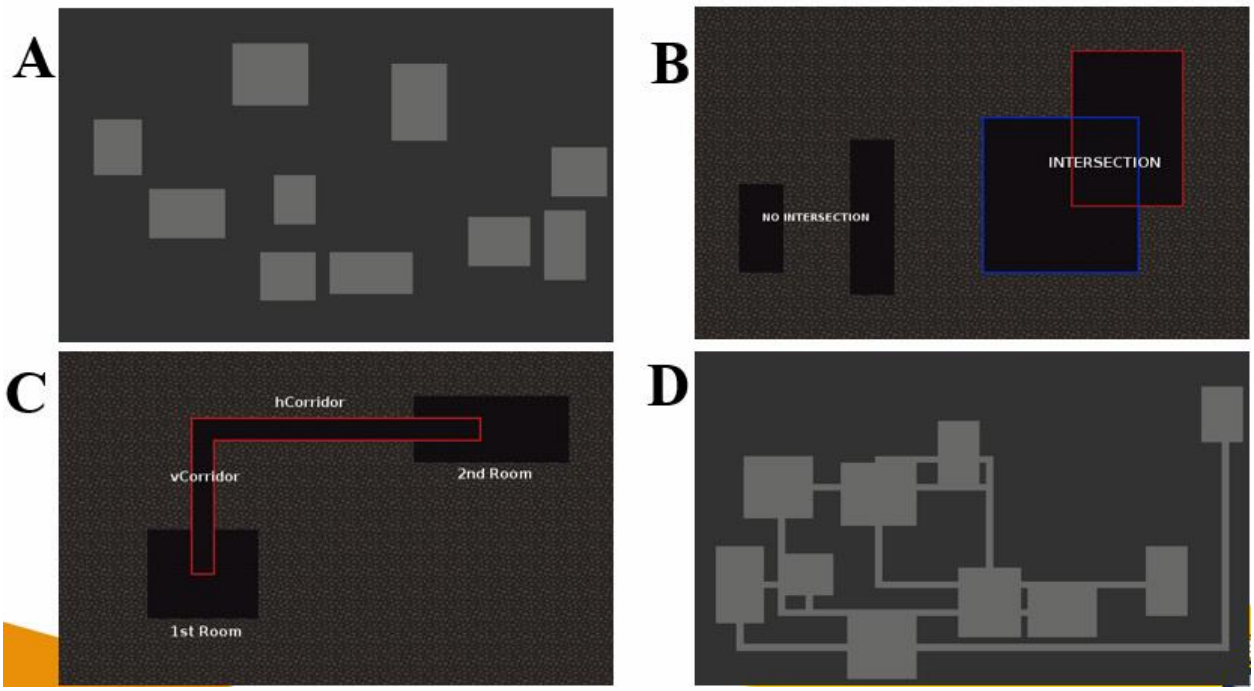
The DoorTrigger class waits for an OnTriggerEnter and then uses the EventManager to call OpenDoorWithID with the doorID specific in the inspector.

As a final note on the Doors example, you can see that the way the doors move has a shadow copy of the door showing where the door will move to. There are three separate doors. The first two move up, but the third moves right. You can see that the door will go wherever the other object is located. This gives a lot of freedom with how your door opens.

6 Procedural Dungeon Map

You can find all the definitions for how objects are spawned in as part of the DungeonManager object with a DungeonManager script, a DungeonGenerator script, and a DungeonObjectGenerator script. All the code and content related to this part of the examples are under the DungeonDemo folder.

6.1 Summary of Room Generation



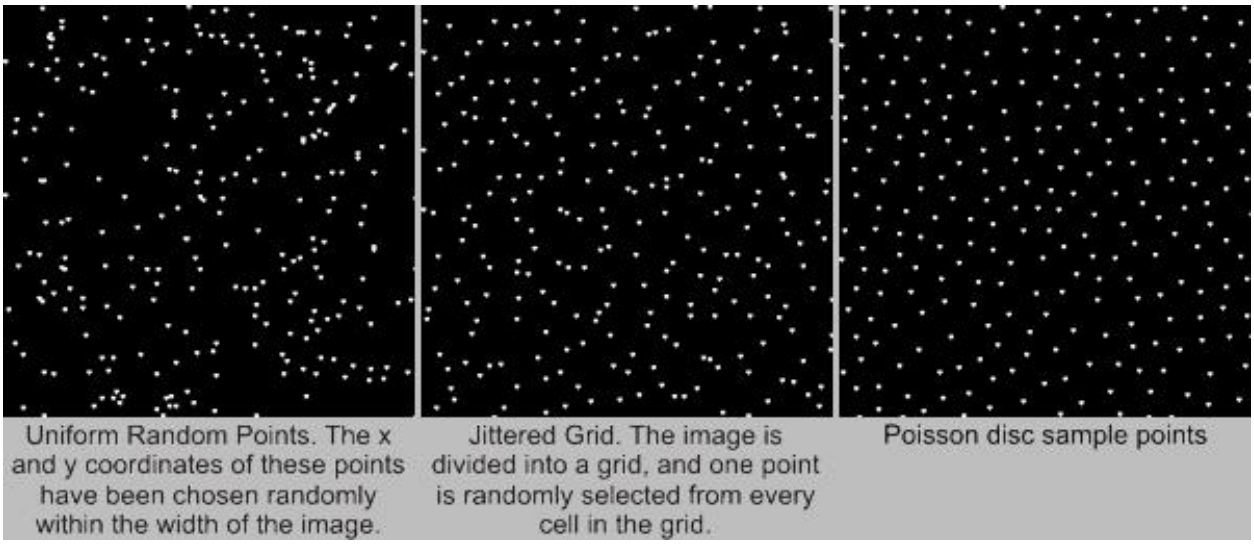
- A: Shows the rooms spawns such that they are not overlapping.
- B: Shows an example of something that is now allowed to happen with rooms intersecting each other.
- C: Shows an example corridor made by adding a vertical corridor to a horizontal corridor.
- D: Shows an example of all the rooms linked together.

This is all managed in the DungeonGenerator class with room generation. The DungeonManager uses references to the DungeonGenerator and DungeonObjectGenerator. The DungeonManager is responsible for spawning the actual objects in.

The DungeonTile objects are on the prefabs spawned in and the sprite is set dynamically via the DungeonManager for each element spawned in.

6.2 Summary of Item Spawning

Below you can see an example of different randomised distributions. You can see that poisson disc sampling has a very even distribution that keeps a minimum distance between objects but not too much. This strategy is combined with a filter to make objects only spawn inside the rooms. And in some cases the corridors.

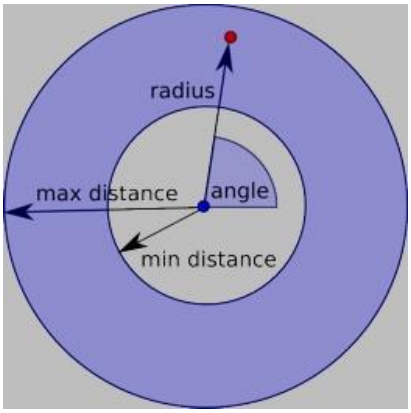


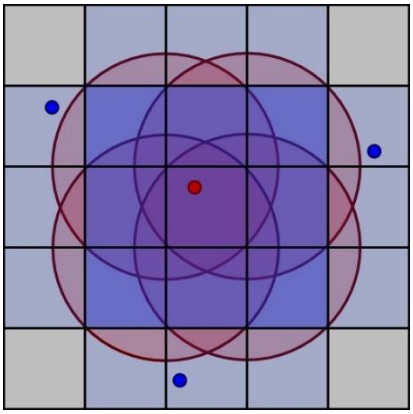
Uniform Random Points. The x and y coordinates of these points have been chosen randomly within the width of the image.

Jittered Grid. The image is divided into a grid, and one point is randomly selected from every cell in the grid.

Poisson disc sample points

The individual points are placed based on the restrictions shown below on the left. And then you can see how the overlapping checks on the right split up the points.





The DungeonObjectGenerator generates a poisson grid and then filters it to match up with the rooms generated in the DungeonGenerator.

7 Chad's Challenge

Was a game developed in 2011 by a group lead by myself along with Brett Mitchell, Kane Stone, and Phill Lavender. It was developed with XNA so it isn't directly related to development with Unity, but it has been included for anyone interested.

Youtube Playthrough Part 1: https://www.youtube.com/watch?v=ESXLwrbM1lM

Youtube Playthrough Part 2: https://www.youtube.com/watch?v=8PLcUNU-Y3U

Chad's Challenge Github URL: https://github.com/Squirrelbear/ChadsChallengeDemoCopy

Note that this only shows levels 1 to 8. For level 9 and 10 see below.

How to pass level 9 and 10

If you are stuck on passing level 9 and 10 as these were not included in the video here is how they may be passed.

Level 9:

1. use 7 blocks ONE AT A TIME to push them into big water straight down centre.
2. use 8th block that doesn't have fire to push it onto red button
3. follow path down to far end and get flippers
4. swim down south get red key
5. swim all way back to start and get coin
6. go back and use red key to open door
7. continue swimming past thieves
8. use green key
9. use portal
10. go up one and then into right portal (top right)
11. collect everything and exit through the one way you can go
12. go south into next portal
13. collect everything and exit
14. go west into next portal
15. collect everything
16. exit and go north into last portal
17. avoid guards and get coins
18. then straight to exit

Level 10:

1. it is possible to use standing on the toggle blocks to pass the level getting across - basically using this way you have to do it very very slowly
2. fast way is that the top 3 separating wall bits between the guards are actually disappearing walls
3. just follow through in order of keys and pass level

The Main Menu

When the user opens Chad's Challenge they will be on the main menu of the game. From this menu they will be able to choose from a number of options including starting a new game, continue a game, starting the level editor, select a level to play, see the help screen, see the credits or exit the game.

Figure 1 : The Main Menu

The Level Selection Screen

When the user opens the level selection screen, they will be able to play any levels they have unlocked or created, when selected these levels will show the high scores, if available. The user will also be able to edit levels they have created but not the levels that come with the game.

Figure 2 : The Level Selection Screen

Playing the Game

When the user begins a new game of Chad's Challenge, they will appear in one of the levels, once inside a level they must navigate their way around the hazards of the level whilst collecting coins. In order to navigate through the level, the user may use either the arrow keys if using the keyboard or left thumb stick on the game pad. The user must collect all of the coins in a level in order to bribe the guard and reach the exit, whilst Chad is collecting the coins he must avoid hazards, including monsters and blocks such as fire. During the game the user may also collect a number of other items which will assist them by allowing them to move freely across other blocks, these items include the fire boots. Chad may also find keys depending upon the level; these keys will be able to open the locked door which matches their colour.

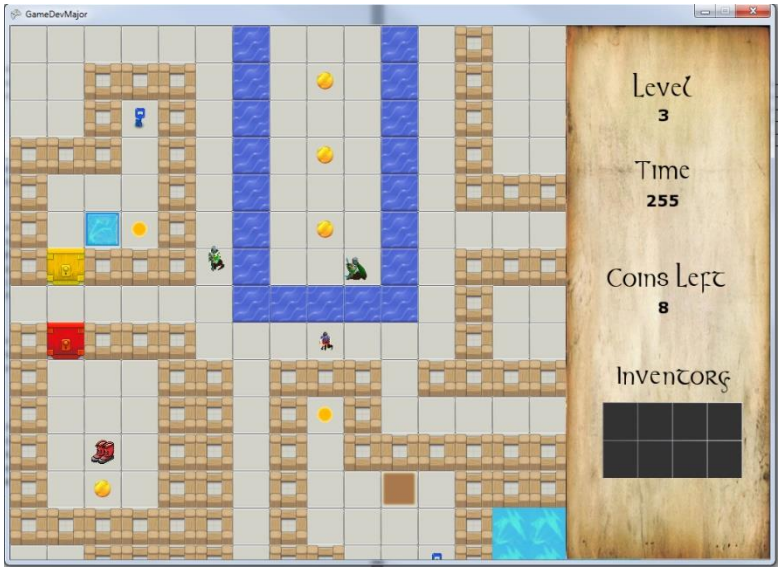


Figure 3 : A screenshot from "The prison" level

Monsters that may be encountered include:

- Assassins: They will mercilessly hunt you. Avoid at as long as possible!
- Moving statues: Keep your eyes peeled you never know where something might come to life.
- Patrolling guards: They will follow their paths guarding their territory. They also move faster than the player does so keep out of their way.
- Rats: These vermin will attack you if you get in their way.
- Imps: These crafty little things will steal keys and run away with them. Their skin colour mutates to show the colour of key that they have. Catching up to the imp will grant the player the key, but the imp will run off to find its next prize.
- Archers: These stationary sentinels stand guard waiting for the order to fire and do not hesitate on command.

There are many other tricks and traps in the game, to not spoil the game it is best that you the player try out everything yourself and discover what each different block can do.

The Editor

When the user opens up the level editor, they will be presented with a blank level which they can add items to in order to challenge their friends. The user will be able to select which block or monster they are laying by using a combination of the Q, W, E, A, S and D keys or the d-pad on the gamepad controller. When using the d-pad on the game controller the up and down directions swaps between the types of things the user can edit, the left and right directions allow the user to change the tile or monster currently selected, the player can then use the shoulder buttons to change the facing. For the keyboard the user uses the W and S keys to swap between the groups of objects, A and S to change the tile or monster and Q and E to select the facing. Once the user has selected the object and facing, they can press the spacebar on the keyboard or the A button on the gamepad which will place the object. When removing a tile or monster the user must use the B key on the gamepad and the delete key on the keyboard. The user will also be able to set the time that the user has to complete each level. When the user wants to test the level, they have created they can press F3.

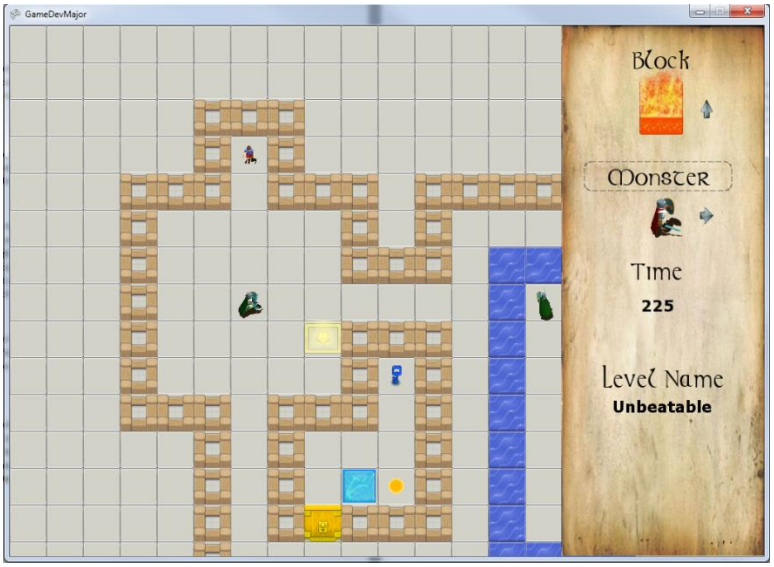


Figure 4 : A screenshot from inside the level editor

Blocks

Most blocks should be apparent in what they do from their image or having interacted during play. There are a number of blocks that require special interaction in the editor to place them.

- The Trap, Monster Spawner (sewer), and Archer: For these tiles you should choose the mode (facing) that you want to place the tile with. Once you press the place object button a prompt will appear at the top of the screen indicating you must select a location for a button. Select another different square to match a button to the object.

Removing either of the objects in any of these pairs will automatically remove both while in the editor mode. Note that the tiles will not appear till both positions are selected.

- Portal: The portal block like the above needs the selection of a second point before graphics will appear. The difference though is that you may chain placing portals. To finish the chain of portals you must select a position where an existing portal graphic is. The message prompt will indicate this to you. Each portal will only appear once each one has a target.

Monsters

Most monsters (except the two movable blocks) can have a direction they start facing. This is all that needs to be set except for the patrolling guard. The patrolling guard when placed allows selection of a number of points that the guard will patrol around. The guards starting location is automatically part of the set of points. The guard will not be shown until all points have been placed and the player has pressed the button as indicated at the top of the screen. It should be noted that the guard will use path finding to find the shortest route to a point.

There are very few limits on what is allowed in the editor mode, so it is up to you to make the most of this and develop incredible levels. These levels you can get your friends to play too and see if anyone is able to pass your best creations.