Tutorial on Unity

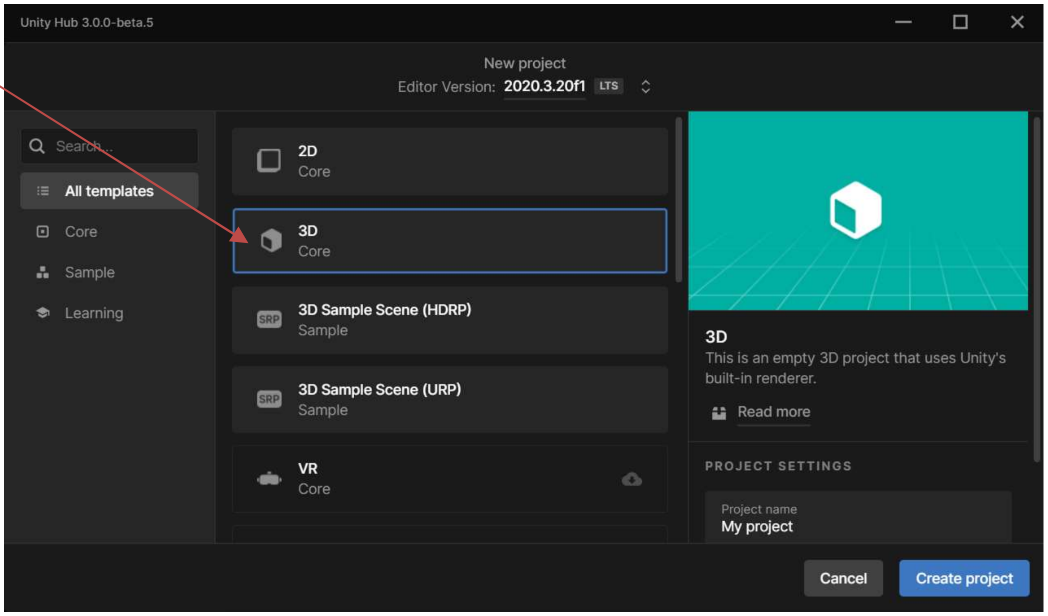
Siu-hang Or

Unity

- http://unity3d.com/
- State of art 3D game engine
- Use C# or UnityScript(Javascript, deprecated in new versions) for scripting
- Need to register to use the free(Personal or student) version
- Paid version with more features on operation side, but as a starting version, free is already enough
- we will develop a simple tank classic game in this tutorial

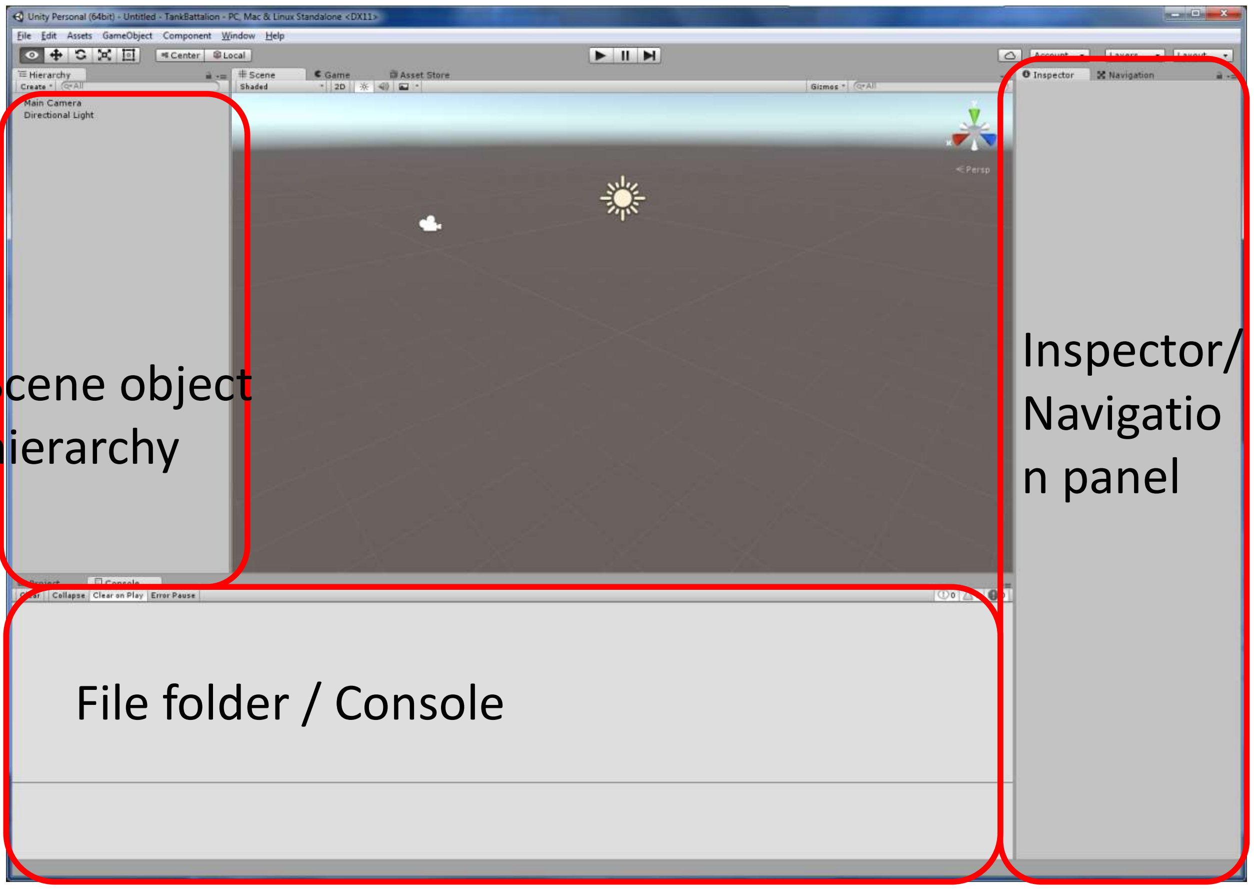
Unity

- First install Unity hub to choose installation (personal or student)
- For student version, you have the option of collaboration, which may be useful in project
- When choosing version to install, you may choose the current version (2022.X.XX)
- In Unity Hub screen, Click "new project"
- Choose this



Create Project

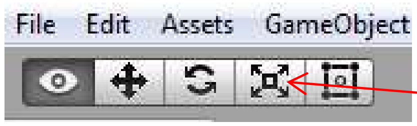
- Type TankBattalion in project name, click "Create Project"



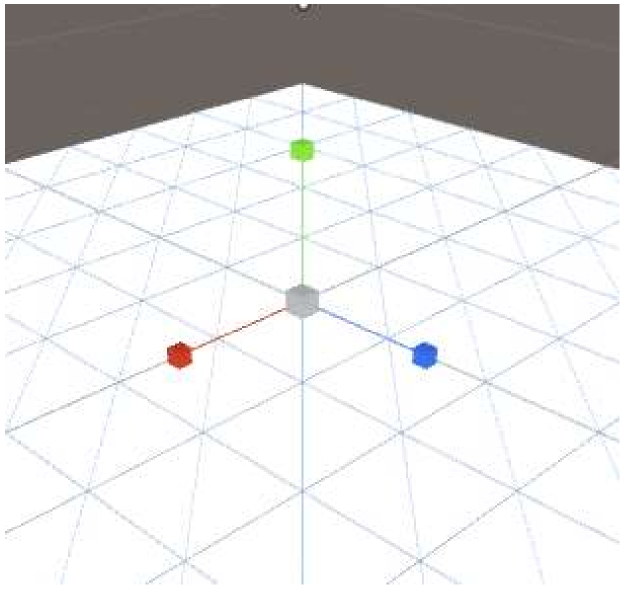
Set up

- We have prepared a package under Blackboard
- Download and extract all the content under the package folder and make them under the Assets folder
- Usually the project folder is under the user folder

Create Level

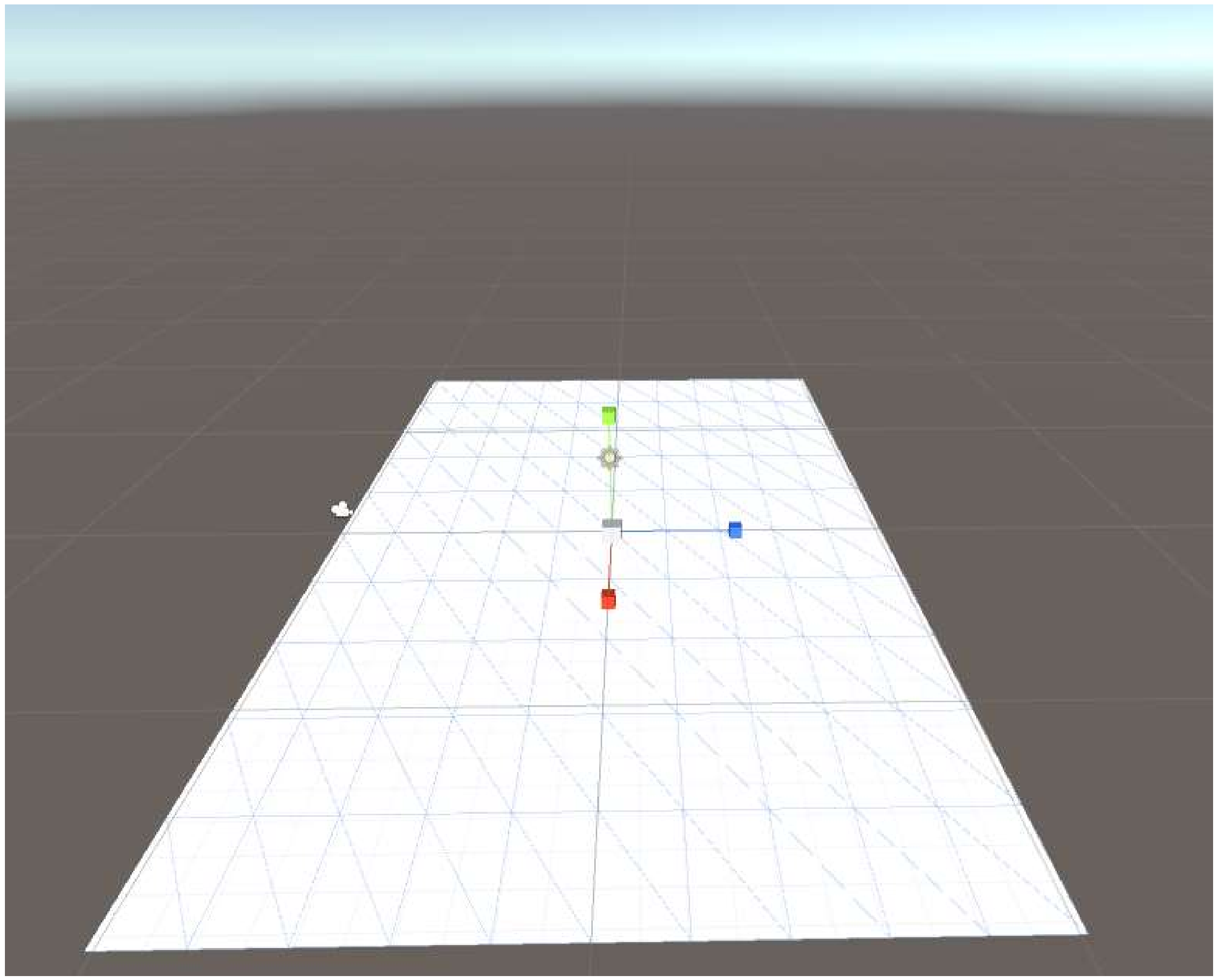
- We will create the ground first
- In main menu, choose “GameObject/3D Object/Plane”
- 

near top left of screen, click on the scale button
- Note the handle should change to like



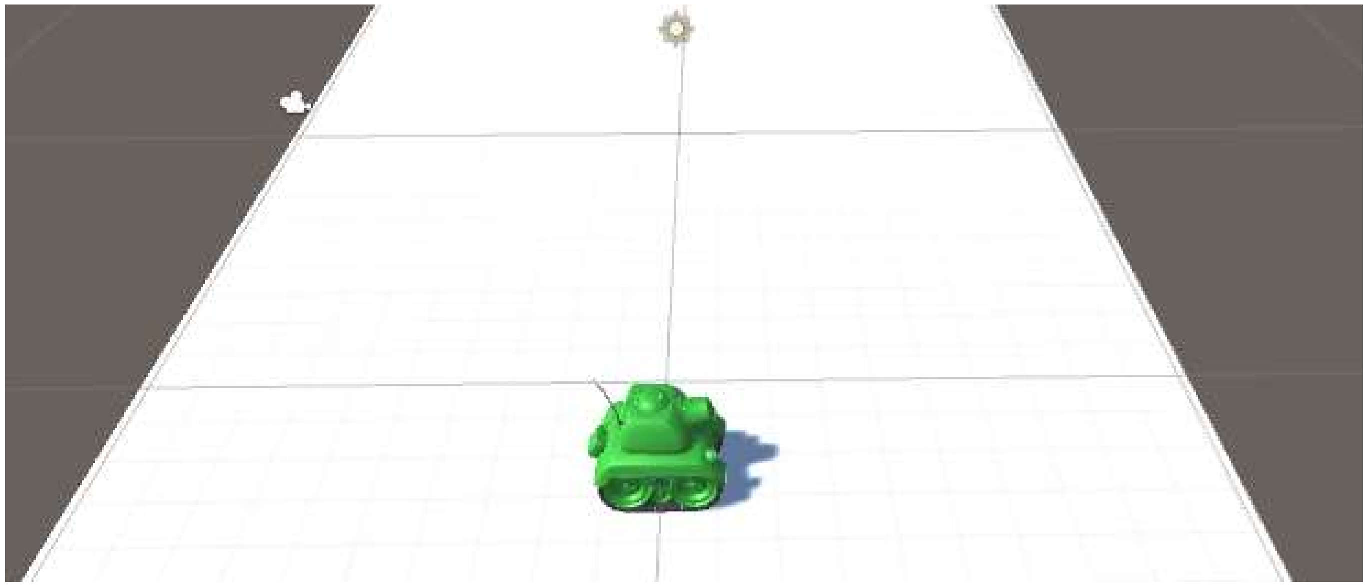
Create Level

- Drag the handle to make the level looks like rectangular shape similar to below



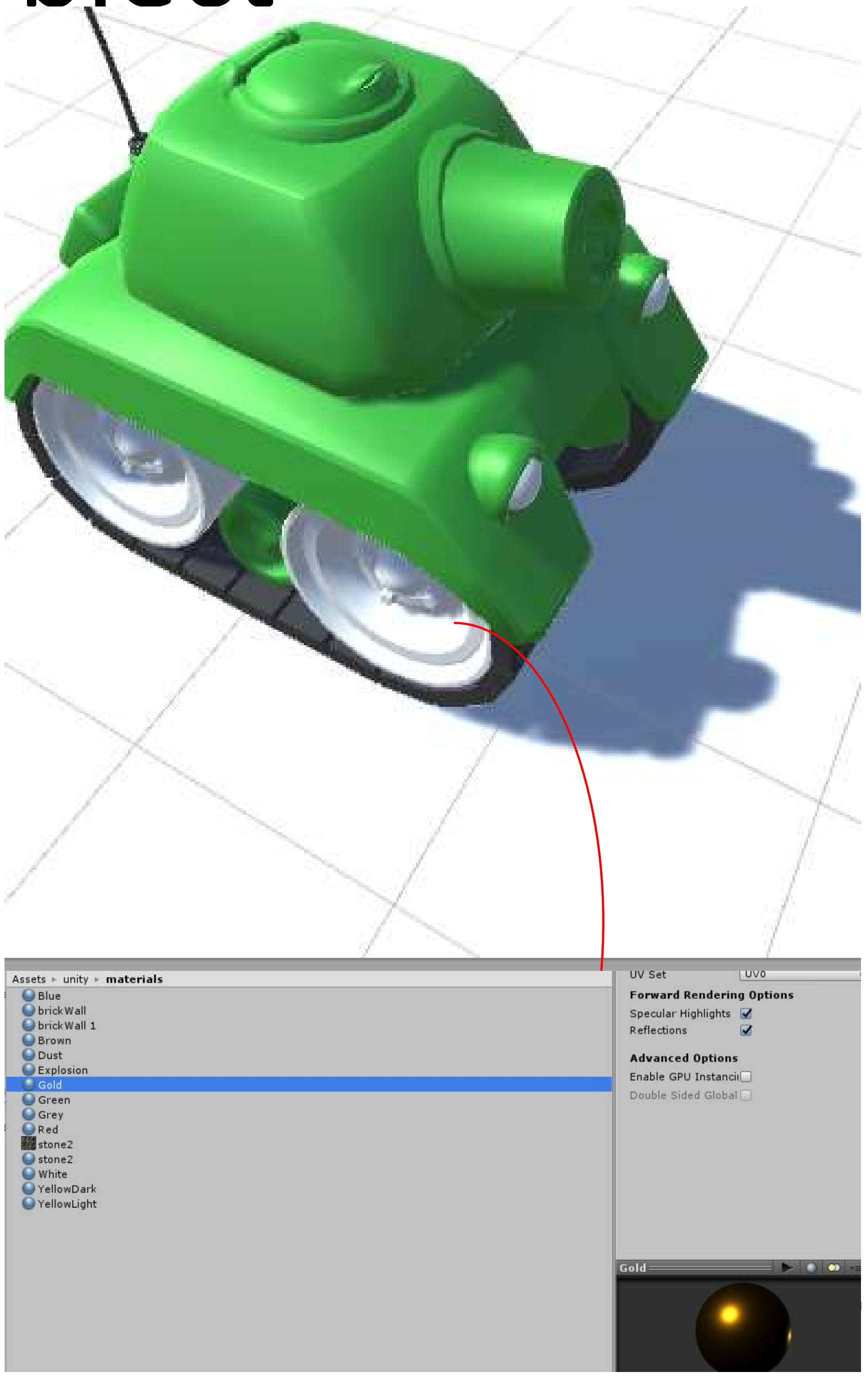
Game Object

- In models folder, drag the tank into the level
- You have created a GameObject of a tank!



Game Object

- GameObject have all the properties needed in a game eg. Interaction, rendering, navigation, physics etc.
- Let us decorate our tank a bit
- Go to Materials folder in project folder
- Drag the TankLights material to the wheels of tank
- This change the materials of tank parts



Assets ▸ unity ▸ materials

- Blue
- brickWall
- brickWall 1
- Brown
- Dust
- Explosion
- Gold
- Green
- Grey
- Red
- stone2
- stone2
- White
- YellowDark
- YellowLight

UV Set UV0

Forward Rendering Options

Specular Highlights ✓

Reflections ✓

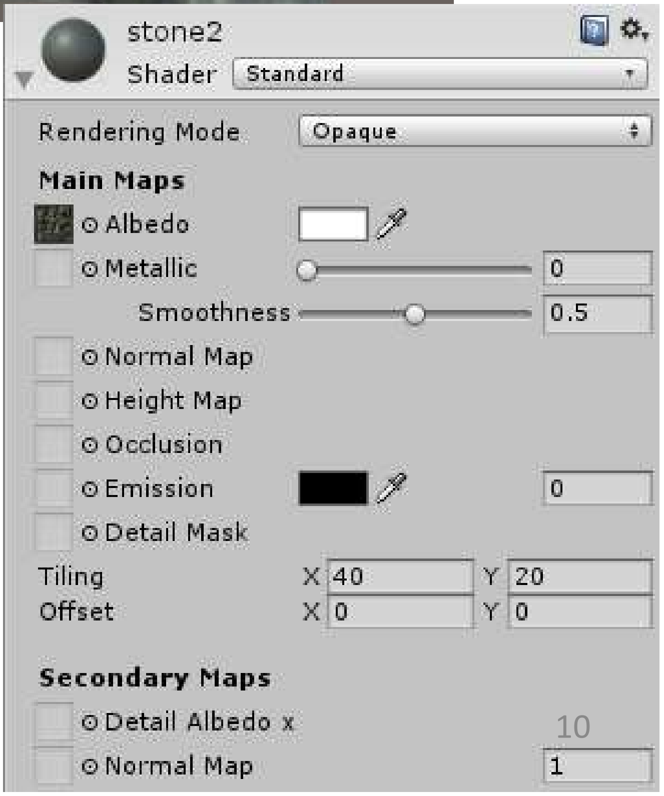
Advanced Options

Enable GPU Instancii ☐

Double Sided Global ☐

Create Geometry

- Drag the stone2 material to the ground plane
- Click the plane to select it
- Under the "Stone2 Shader", adjust the "Tiling" X & Y values to make it look good



stone2
Shader Standard

Rendering Mode Opaque

Main Maps

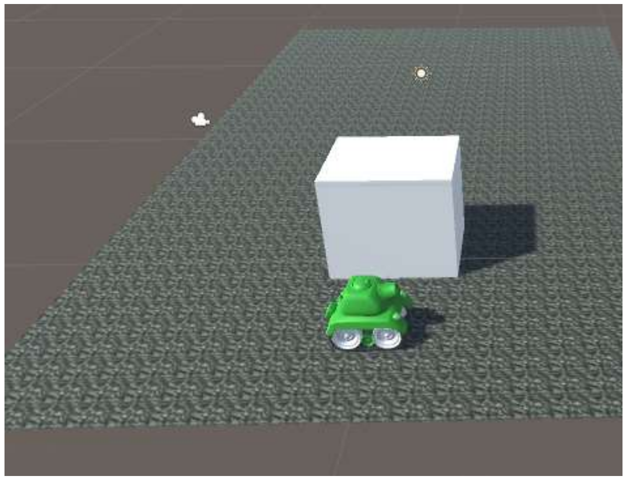
Albedo		
Metallic		0
Smoothness		0.5
Normal Map		
Height Map		
Occlusion		
Emission		0
Detail Mask		
Tiling	X 40	Y 20
Offset	X 0	Y 0

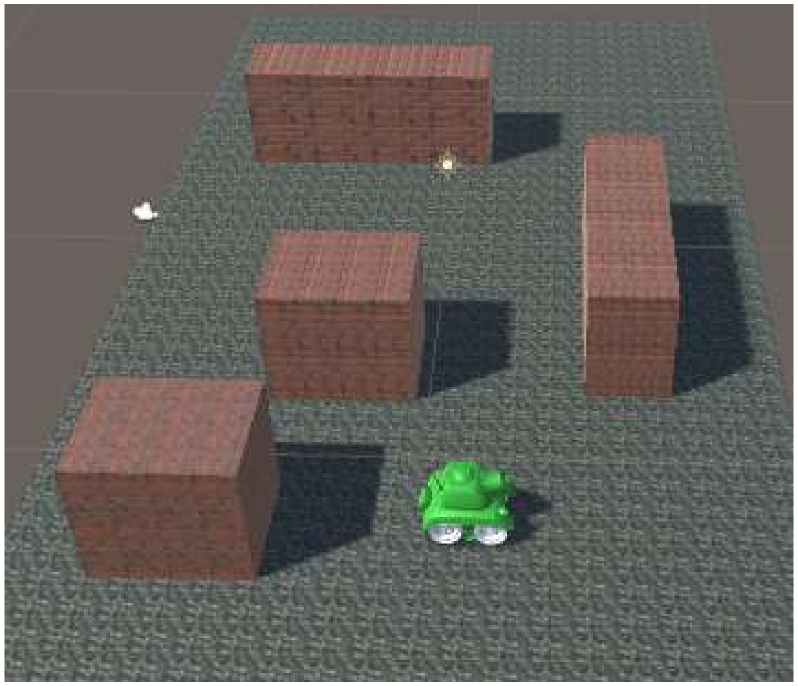
Secondary Maps

Detail Albedo x	
Normal Map	1

Create Geometry

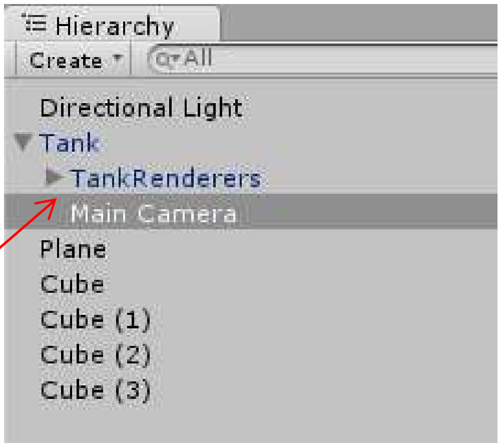
- Create a cube with size similar to right
- Click "Assets/Import New Assets.."
- Browse to the downloaded package and choose "BrickWall.png"
- Drag the created brickwall material to the 3D cube
- Adjust the tiling factor to make it looks good



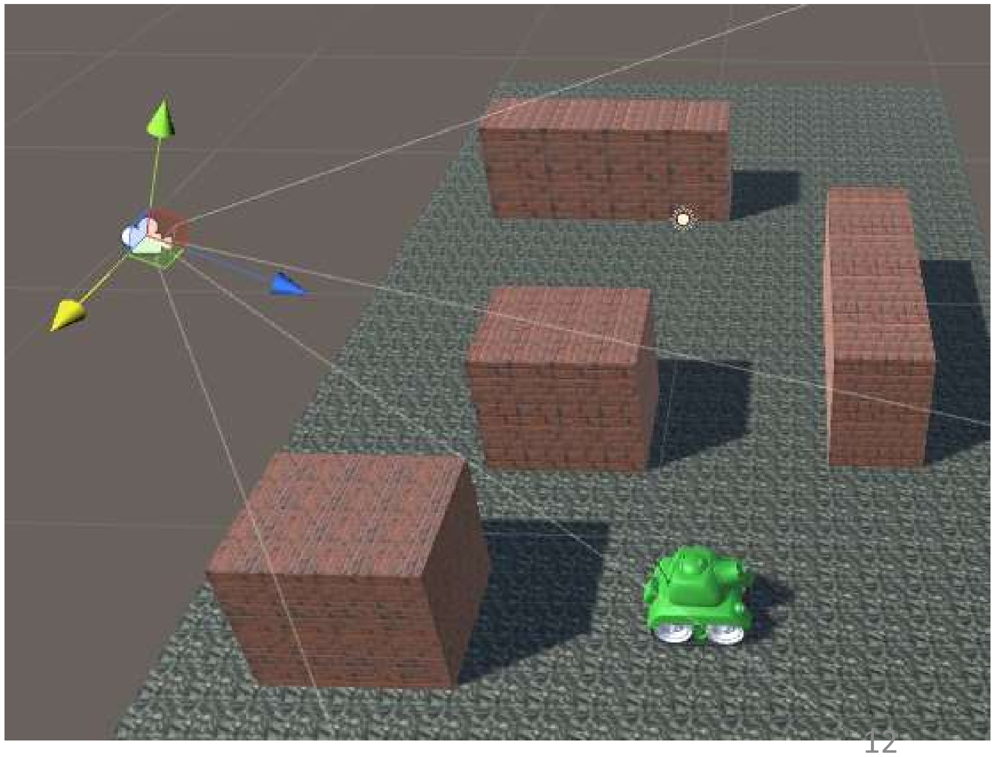


Player Control

- We wish to set up control to our tank
- We want the camera to follow our tank player
- Easy to do in Unity!
- Just drag the "Main Camera" in Hierarchy to under the tank
- Now adjust the camera position and rotation in the scene so that it overlook the tank

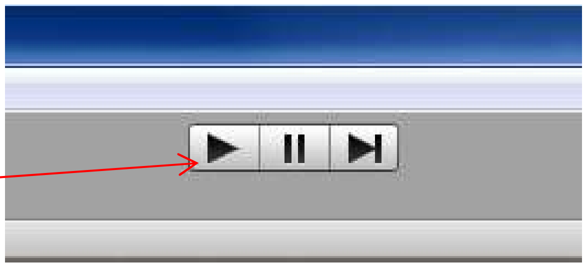


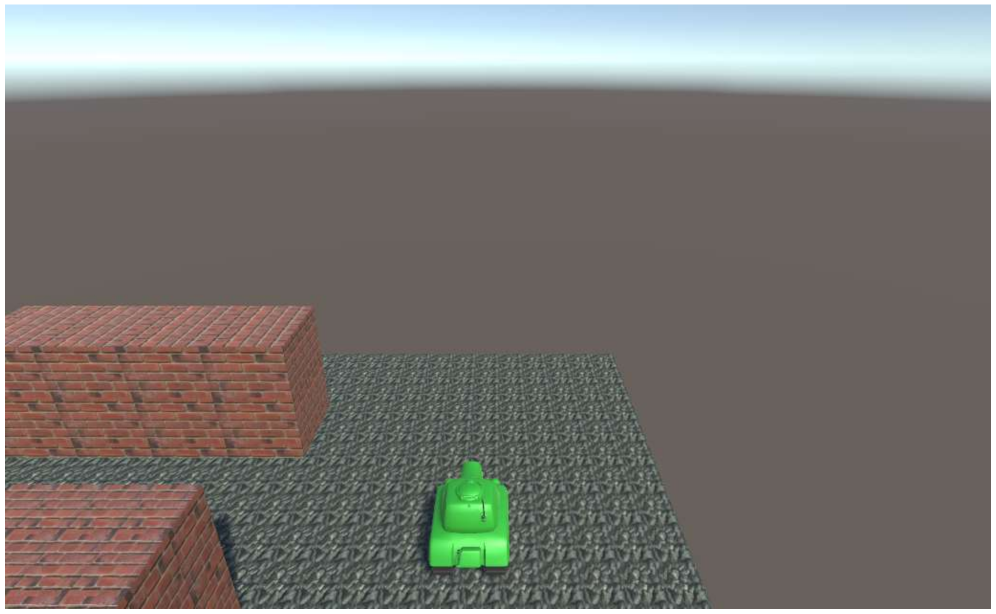
Directional Light
Tank
TankRenderers
Main Camera
Plane
Cube
Cube (1)
Cube (2)
Cube (3)



Player Control

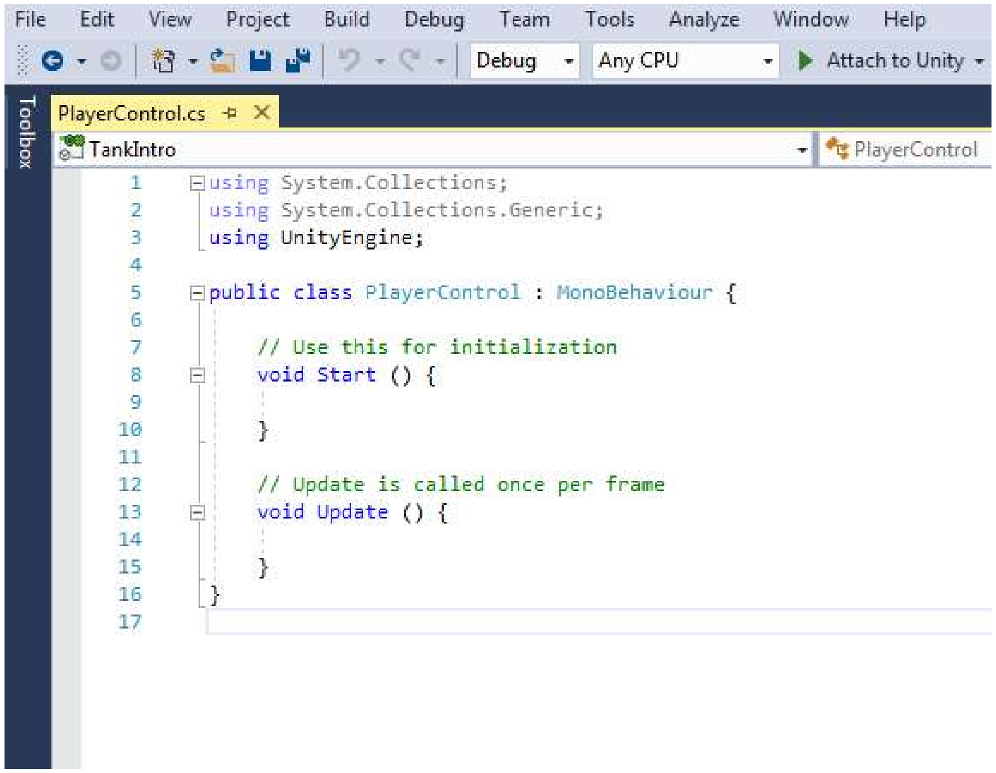
- Now click on the play button on top middle screen to check the play scene
- But the tank can't move!
- Let's add interactivity.
- We use Script to control movement in Unity
- Select "Assets/Create/C# script"





Player Control

- A file appear in Project panel, type "PlayerControl" to name it
- Double click it to open the script editor
- Start is the function to control what to do right at start
- Update corresponds to things to be done before **each** rendering frame



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerControl : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

You may choose your favorite script editor by "Edit/Preferences/External Tools" to choose "External Script editor"
Now by default it is c#, so you need to install MS Visual Studio

Player Control

- Type the following statements under the Update function

```
float forwardAmount = Input.GetAxis("Vertical") * forwardRate;
float turnForce = Input.GetAxis("Horizontal") * turnRate;
transform.Rotate(0, turnForce, 0);
transform.position += transform.forward * forwardAmount *
Time.deltaTime;
```

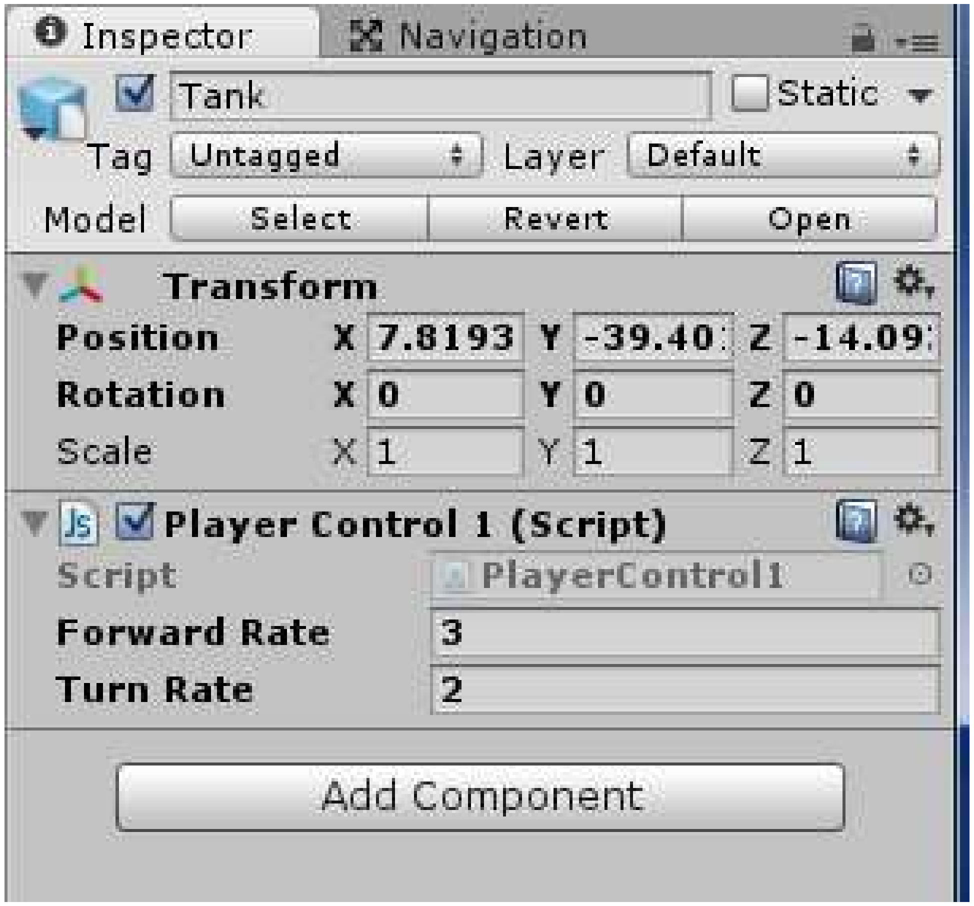
Add the following lines to global area of the class

```
public float forwardRate = 3.0F;
public float turnRate  = 2.0F;
```

- Save the file and return to Unity editor

Player Control

- Select the tank
- In the Inspector pane, click "Add Component" button
- Choose "Scripts/PlayerControl.cs"
- Your tank Inspector now should be like on right
- Note the variables in **global** area now can be **adjusted**
- Play around with the tank!



Tank						
Tag	Untagged	Layer	Default			
Model	Select	Revert	Open			
Transform						
Position	X	7.8193	Y	-39.40	Z	-14.09
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1
Player Control 1 (Script)						
Script	PlayerControl1					
Forward Rate	3					
Turn Rate	2					

You can choose your key mapping by "Edit/Project Setting/Input" and choose different keys

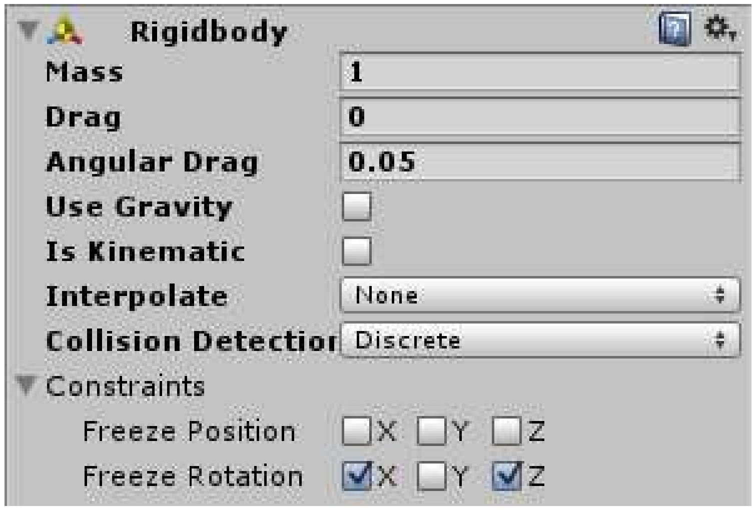
Player Control

Dissection of code

- Adding script file to the tank object means the interaction of the tank will be under script control
- The scripts in update function means we control the tank in every frame according to the instructions there
- Input.GetAxis will read the keyboard input as defined in Input object (default one, vertical refers to up/down arrow keys, horizontal for left/right arrow key)
- transform is the rotation+translation description of the object itself (tank)
- transform.forward is the vector of (0, 0, 1)
- Time.deltaTime is the time elapsed during this frame to now

Physics & Collision detection

- Our tank can pass through all the blocks, which is not realistic
- Click the tank and Add a "Physics/Rigidbody" to it
- In the Constraints section, set the freeze rotation on for X and Z direction, uncheck "Use gravity"
- Add a "Physics/Box Collider" to the tank, click on "Edit Collider" to edit the bounding box to fit the tank
- Green line is the collider
- Test your collider!

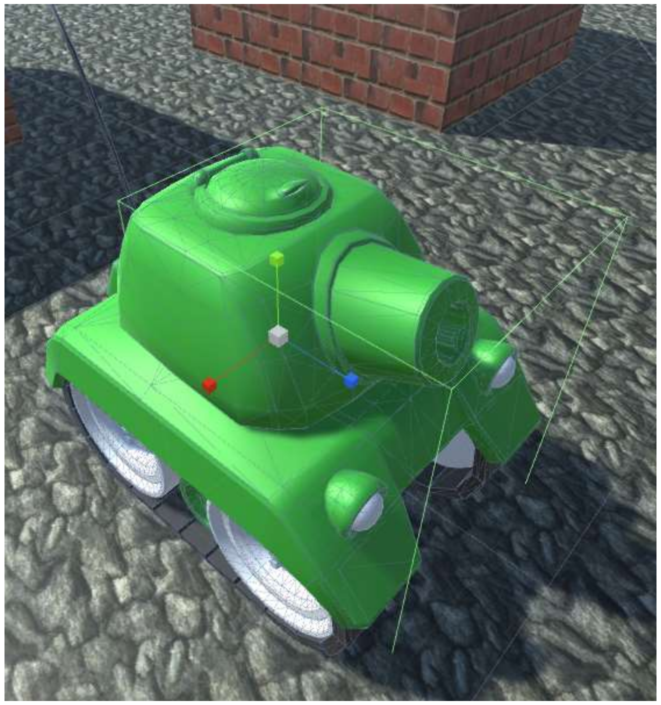


Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete

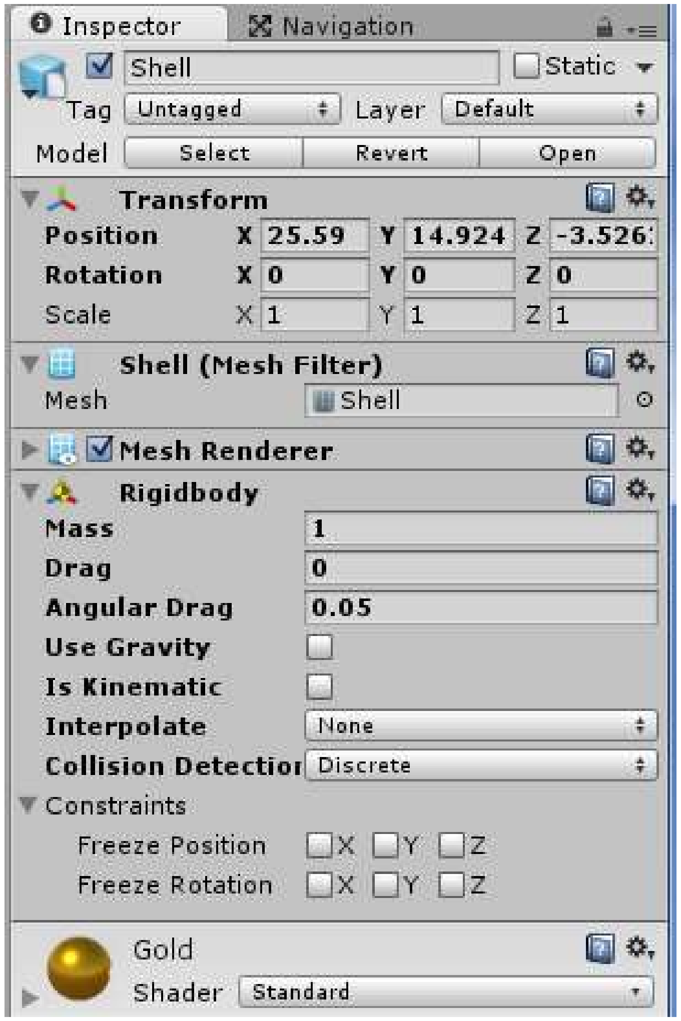
Constraints

	X	Y	Z
Freeze Position	☐	☐	☐
Freeze Rotation	✓	☐	✓



Fire!

- We want to enable our tank to fire
- Browse to models under Assets folder
- Drag "Shell.fbx" into a position outside the plane
- Add a Rigidbody component to the Shell, uncheck "Use Gravity"
- Add also a collider, say capsule or box to the shell
- next to create the script to control firing
- Choose "Assets/Create/C# Script" and name it "shoot"



Shell ☐ Static
Tag Untagged Layer Default

Transform
Position X 25.59 Y 14.924 Z -3.526
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Shell (Mesh Filter)
Mesh Shell

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ☐
Is Kinematic ☐
Interpolate None
Collision Detection Discrete
Freeze Position ☐X ☐Y ☐Z
Freeze Rotation ☐X ☐Y ☐Z

Gold
Shader Standard

Fire

- Modify your shoot.cs
- Add global variables in shoot.cs

```
public Rigidbody prefabBullet;
public float shootForce;
public Transform shootPosition;
```

- Then add statements below to Update method

```
if (Input.GetButtonDown(“Fire1")) {
  Rigidbody instanceBullet =
      (Rigidbody)Instantiate(prefabBullet,
      transform.position+ transform.forward * 1.5F,
shootPosition.rotation);
  instanceBullet.GetComponent<Rigidbody>().
      AddForce(shootPosition.forward * shootForce);
}
```

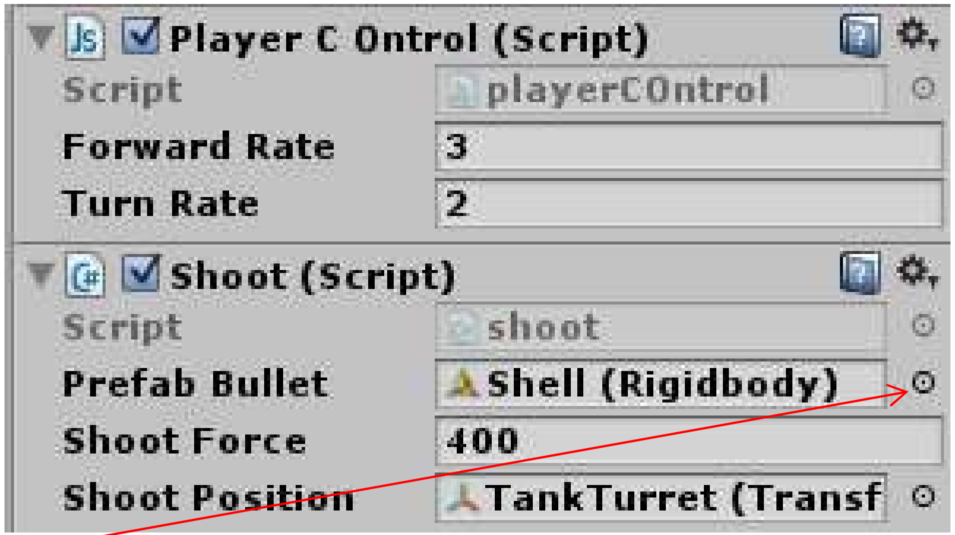
Fire

Dissection of Code

- “Fire1” button corresponds to left-Ctl key
- Instantiate is an inherited method from Object which used to create an instance of another object
- AddForce method will cast a force on the object to make it move
- Note that using force is the preferred method in interacting with objects in game world
- In general modifying the position, velocity (or angular correspondences) of an object is not recommended

Fire!

- Back to Unity editor, select the tank
- Add “shoot.cs” to tank
- Configure the parameters by clicking on the circle button on right
- Note you should set your shell object to have a *rigidbody* in previous steps
- Shoot position is just any point of the tank, we choose Tank Turret here



Player C Ontrol (Script)

Script	playerCOntrol
Forward Rate	3
Turn Rate	2

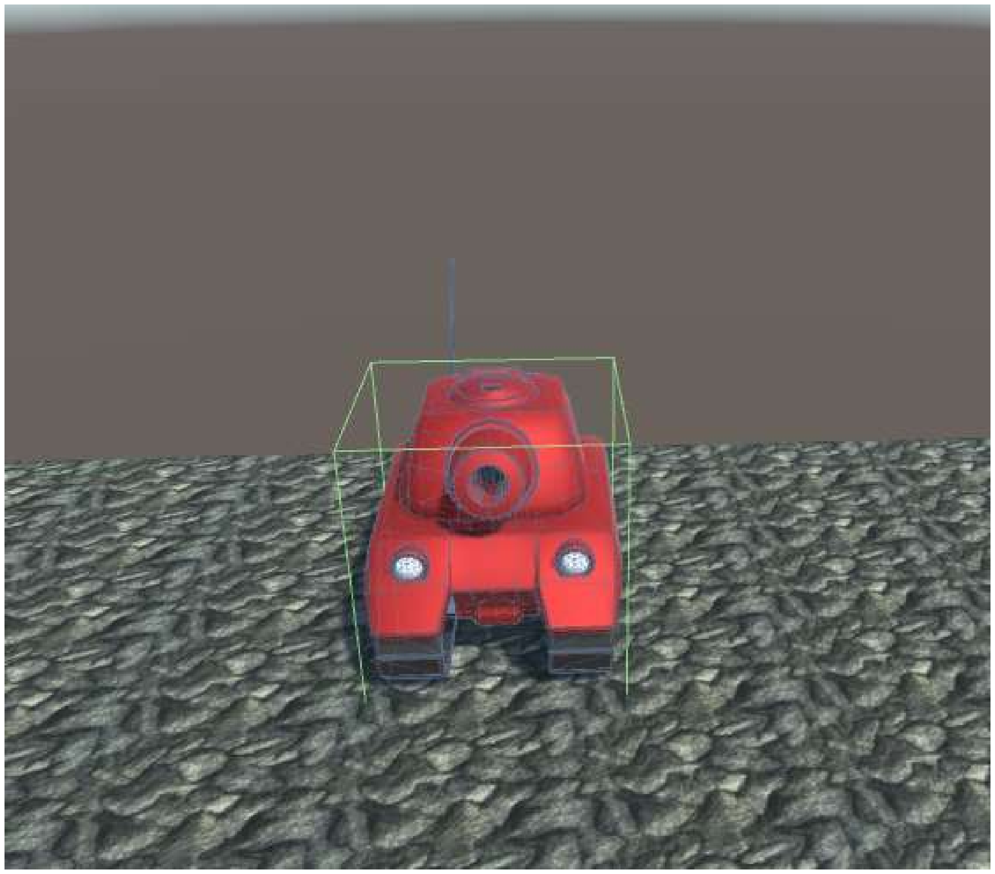
Shoot (Script)

Script	shoot
Prefab Bullet	Shell (Rigidbody)
Shoot Force	400
Shoot Position	TankTurret (Transf

Play to see the shell fire in action!

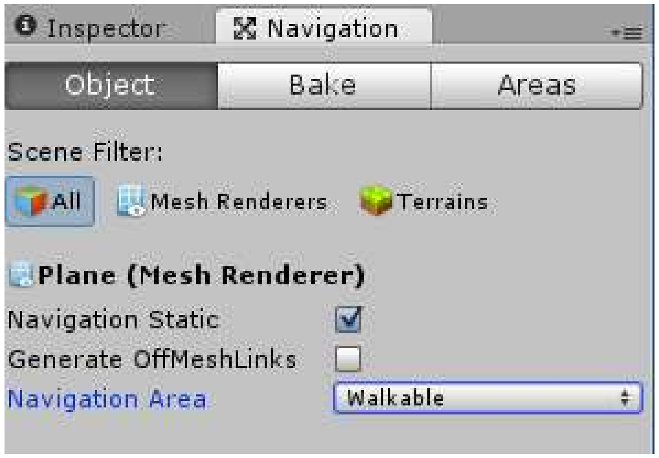
Add Enemy

- Add an enemy tank by dragging the tank model into level
- Change its color for differentiation
- Add a box collider to tank
- Now we wish enemy tank can navigate freely on the level
- Choose "Window/AI/Navigation"

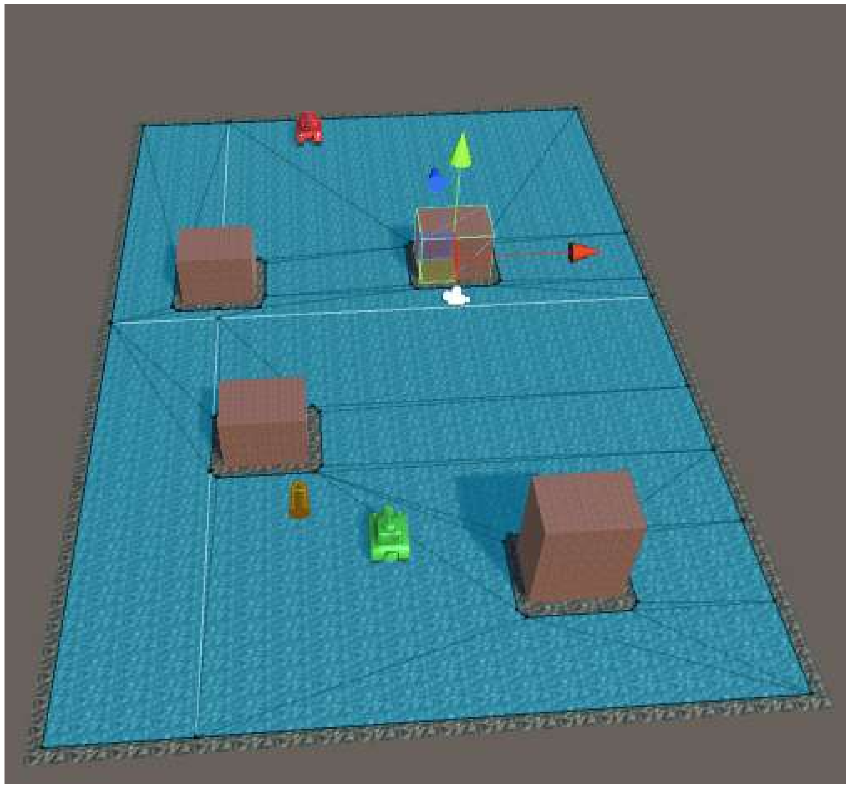


Add Enemy

- The Navigation tab should now be shown over Inspector tab
- Select the level floor and check the "Navigation Static" and select it to be "Walkable"
- Select the other blocks on the level and make them "Navigation Static" but "Not walkable"
- Click the Bake tab and click "Bake" button on lower right, your level should now be similar to right
- You have successfully created the Navigation Mesh!

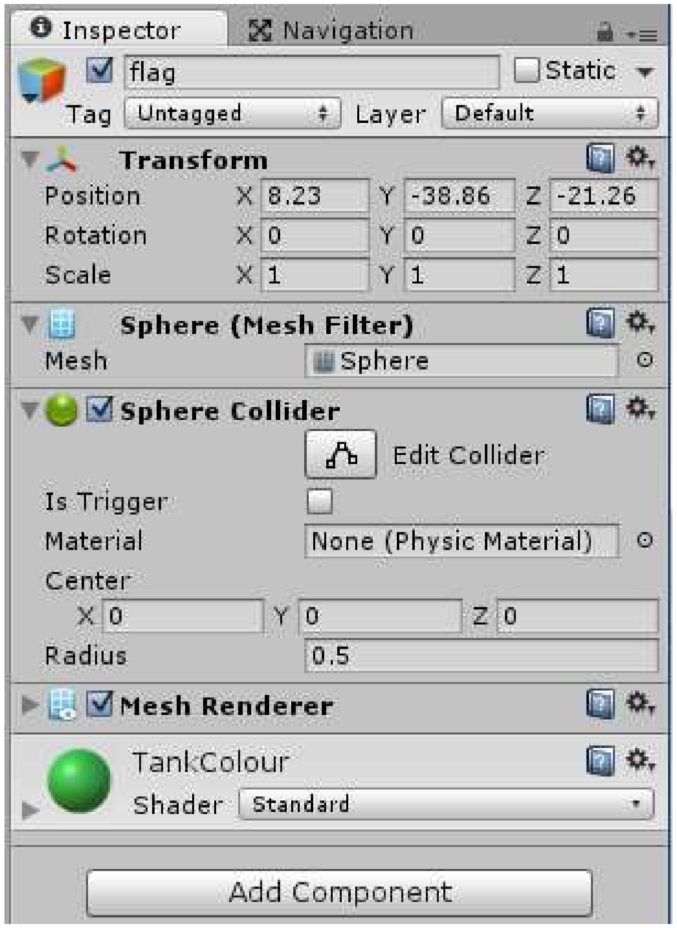


Plane (Mesh Renderer)
Navigation Static ✓
Generate OffMeshLinks ☐
Navigation Area: Walkable



Add Enemy

- Let's create something that your tank try to protect
- Create a sphere and placed it at lower middle position just behind our tank
- Change its name to "flag" in Inspector pane
- Then select the enemy tank
- Add "component/navigation/ NavMeshAgent"



flag
Static
Tag Untagged
Layer Default

Transform
Position X 8.23 Y -38.86 Z -21.26
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Sphere (Mesh Filter)
Mesh Sphere

Sphere Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Radius 0.5

Mesh Renderer

TankColour
Shader Standard

Add Enemy

- Create an asset of C# Script and name it enemyTank
- Add the following declaration

```
public Transform goal;
public int lifePoint = 30;
private UnityEngine.AI.NavMeshAgent agent;
```

- Note agent is private so as to achieve encapsulation
- NavMeshAgent will be able to construct a path from initial position to destination
- agent need be properly initialized to enable navigation

Add Enemy

- Set the values for the script:
- Set “flag” as “goal” field in editor and add the following code to start()

```
agent = GetComponent<UnityEngine.AI.NavMeshAgent>();
agent.destination = goal.position;
```

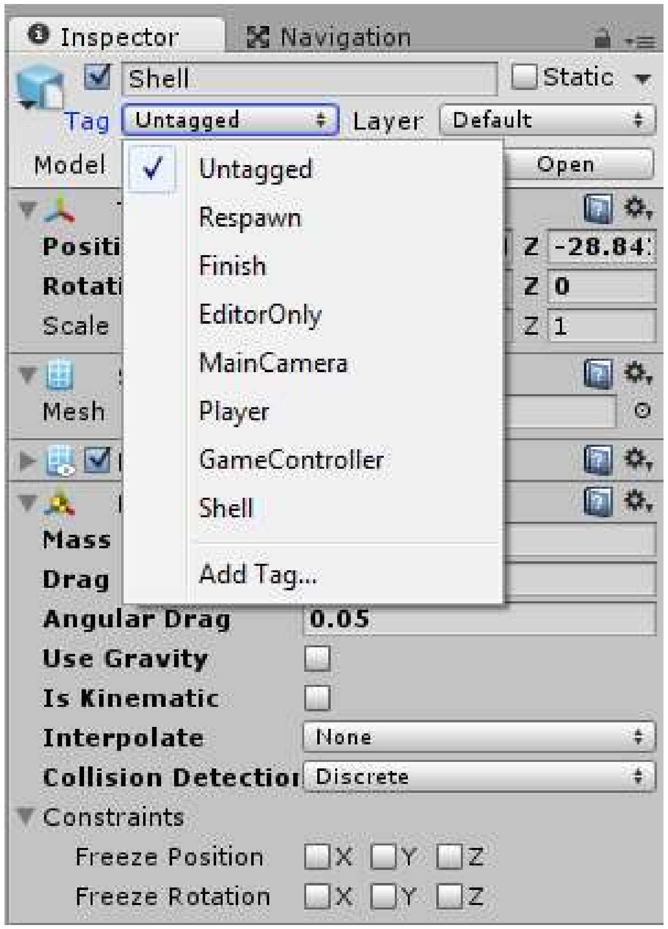
- GetComponent is inherited method of object
- As we already created the NavMeshAgent under enemy, thus first line simply get that into agent
- Now you can test play the level to see enemy tank in action

Let’s Get Tank Killed!

- As a simple action game, we wish the enemy can be killed
- Remember we grant life point to tank
- How to make it get to zero?
- Our tank can fire a shell and if you test it, the shell can really hit enemy tank, but it didn’t get killed
- We have to detect the collision between shell and tank, and reduce life point

Let’s Get Tank Killed!

- select shell in editor and under Inspector, click on Tag field
- Choose “Add Tag” and in the textbox, enter “Shell”
- This add a new tag called “Shell” which identify a new class of objects
- Select Shell in Hierarchy again and select “Shell” in tag field
- Now our shell object will be tagged as “Shell”



Shell
Static
Tag Untagged
Layer Default
Model
Position Z -28.84
Rotation Z 0
Scale Z 1
Mesh
Mass
Drag
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z

Let’s Get Tank Killed!

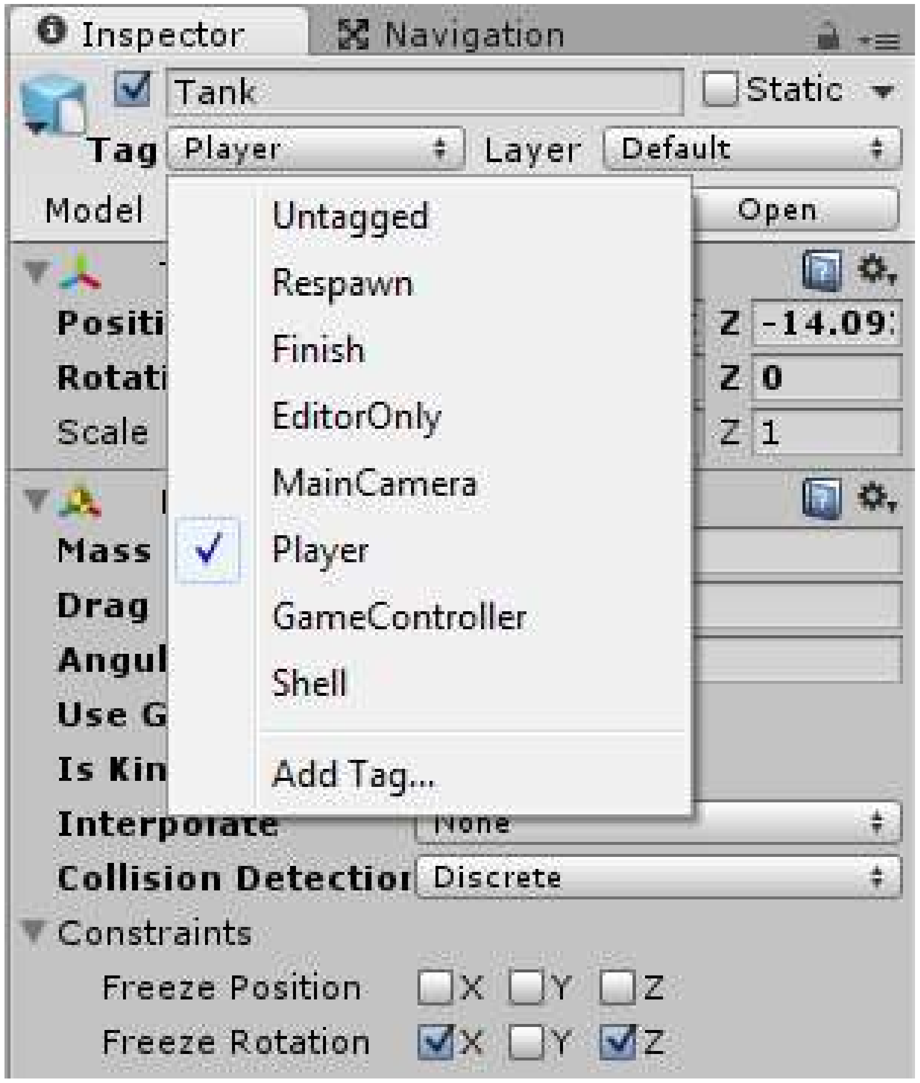
- Edit the enemyTank script again by adding the following function

```
private void OnCollisionEnter(Collision otherObj)
  {
     if (otherObj.collider.tag == "Shell")
     {
        lifePoint -= 10;
        if (lifePoint <= 0) Destroy(gameObject, 0.5F);
     }
  }
```

- The code instruct the tank to check whether it collide with “Shell” or not during collision detection
- If its life point drop to zero, remove it from system with a 0.5 second delay
- Now have some fun in destroying enemy!

Attack Mode

- It's not challenging if the enemy can't attack
- Granting AI to game object is not an easy job!
- First we want it to be aware of its enemy
- Select player tank and set its Tag filed to "Player"



Attack Mode

- Add the following code to enemyTank global area

```
public float scanRange = 10.0F;
private int state;
```

- “scanRange” defined the radius of scanning
- Note we make “state” variable private as it is only used internally in the script
- It indicate whether the tank is in attack mode or not

Attack Mode

- Add the following code into the **Update** function

```
GameObject enemy;
 Vector3 heading;

 enemy = GameObject.FindGameObjectWithTag("Player");
 heading = enemy.transform.position - transform.position;
 if (state == 0) {
     if (heading.sqrMagnitude < scanRange * scanRange)
     {
         state = 1;
         agent.isStopped = true;
         Debug.Log("player within range");
     } }
```

Attack Mode

- The code can only locate player within range
- After entering into attack mode, nothing was done
- Should initiating attack! But we have a problem:
- Update method will do same thing in every frame

- But attack action is a series of actions – align tank to opponent, fire a shell, wait ,..
- We need a function which can carry on over many frames
- Meet **Coroutine** – a function which can suspend its execution until a given special instruction (typically wait instruction) finishes

Attack Mode

- Add the following function to enemyTank.c#

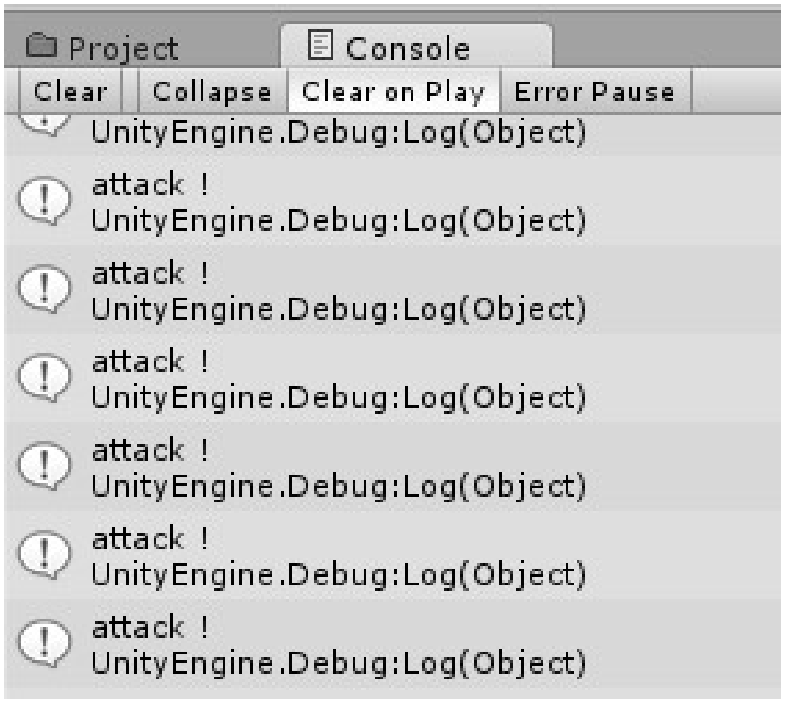
```
IEnumerator attackOrMove()
  {  GameObject enemy;

      while (true)  {
         if (state == 0) {
            yield return new WaitForSeconds(0.1f);
         }
         else  {
}
      }
  }
```

- And add this instruction at the end of Start function in enemyTank.c#

```
StartCoroutine("attackOrMove");
```

Attack Mode

- Now return to Unity editor and run your game
- When your tank try to get closer to enemy tank, you should be able to see messages scrolling up as on right
- This indicate that the enemy tank spot the player
- Note the “yield .. Wait..” statement delay the execution of next round in attackOrMove



```
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
attack !
UnityEngine.Debug:Log(Object)
```

We can replace our attack action with the “attack !” log

Attack Mode

- We wish the enemy tank to target at the player continuously in attack mode
- Modify the Update function as follow

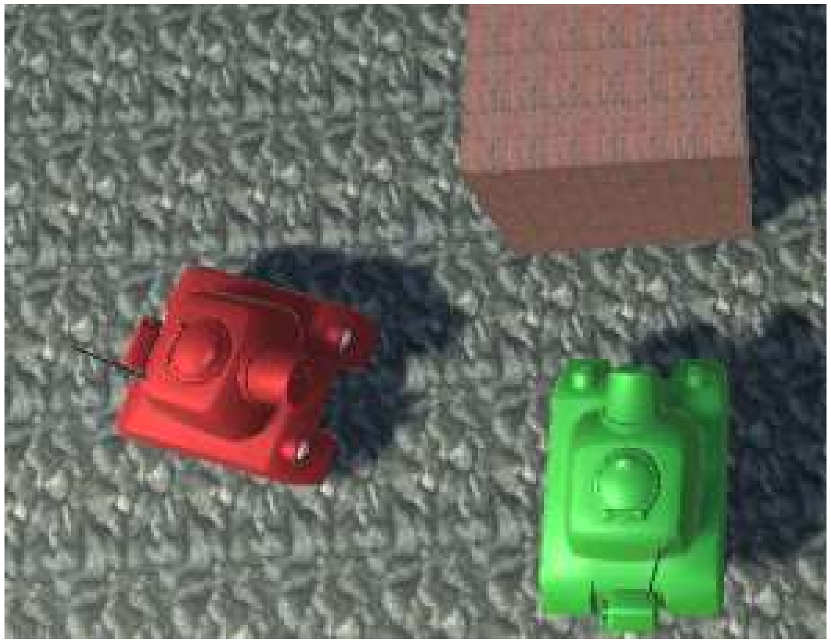
```
 if (state == 0) {
   if (heading.sqrMagnitude < scanRange * scanRange) {
     state = 1;
     agent.isStopped = true;
   }
 }
 else {
  Quaternion rotation =
Quaternion.LookRotation(enemy.transform.position -
transform.position);
  transform.rotation =
Quaternion.Slerp(transform.rotation, rotation,
Time.deltaTime * rotationDamping);
 }
```

- Also add the following line in global area

```
public float rotationDamping = 6.0f;
```

Attack Mode

- Now return to Unity editor and run your game
- When your tank try to get closer to enemy tank, you should see the tank rotate to aim at player tank



We can replace our attack action with the "attack !" log

Code Dissection

- Quaternion is a representation of rotation similar to matrix
- Slerp is for interpolation of rotation using Quaternion

Attack Fire!

- It's time to make the enemy tank shoot at player!
- Replace the code in *else* part of AttackOrMove function with the following command:

```
 enemy = GameObject.FindWithTag("Player");
 transform.LookAt(enemy.transform.position);
 Rigidbody instanceBullet =    Instantiate(prefabBullet,
shootPosition.position + shootPosition.forward * 2.5f,
shootPosition.rotation);
 instanceBullet.GetComponent< Rigidbody >
().AddForce(shootPosition.forward * shootForce);
 yield return new WaitForSeconds(2.0f);
```

- Add the following line in global area

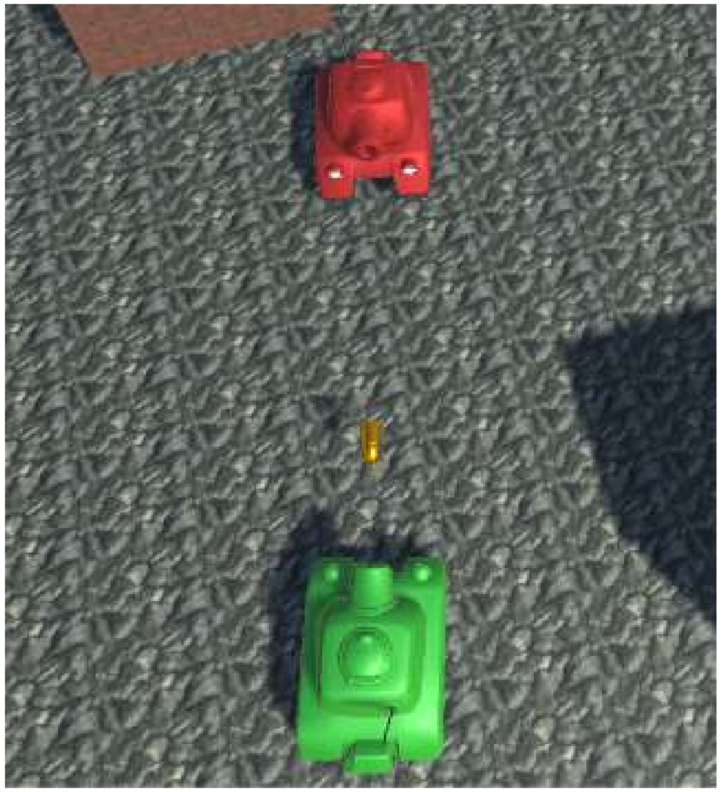
```
    public Rigidbody prefabBullet;
    public float shootForce;
    public Transform shootPosition;
```

Attack Mode

- Remember to do proper set up for parameters in enemy Tank (recall how to make our player tank shoot?)
- Now return to Unity editor and run your game

Code Dissection

- Basically same as that making player


Enemy tank fire at player

Spawning Enemy

- Now our level got only one tank to attack our player
- We want it like arcade game in that a number of enemies are on the level
- Auto spawn enemy is common
- Let’s write scripts to make spawning of enemies!
- First save all our current progress
- Create a GameObject called “GameManager”
- Create a script called “GameManager.cs”

Spawning Enemy

- In GameManager script add the following global declaration

```
public GameObject spawnTank;
public Transform[] spawnPt;
public int enemyTotal = 10;
```

- Modify Start as below

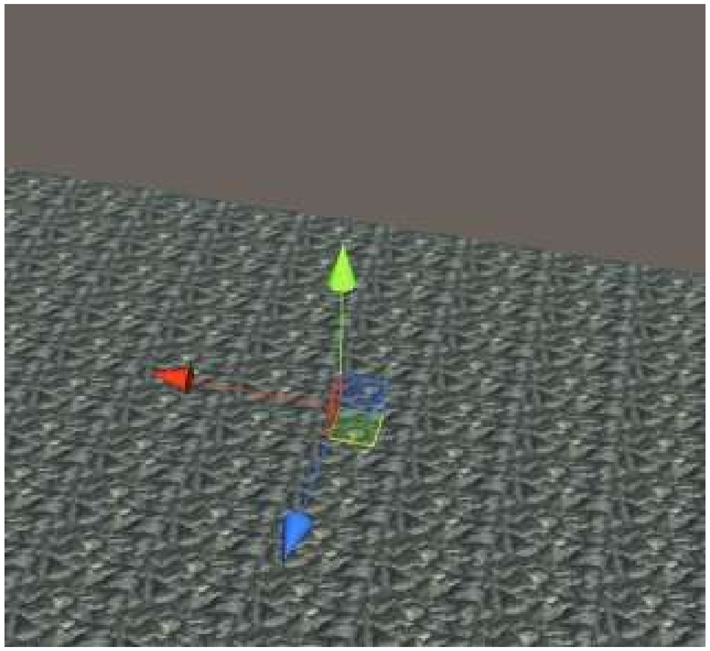
```
void Start () {
        StartCoroutine("spawnEnemy");
    }
IEnumerator spawnEnemy()    {
        int pt;
        for (int i = 0; i < enemyTotal; i++) {
            pt = (int)Random.Range(0.0f, 2.99f);
            var instanceTank = Instantiate(spawnTank, 
spawnPt[pt].position, spawnPt[pt].rotation);
            yield return new WaitForSeconds(10.0f);
    }
```

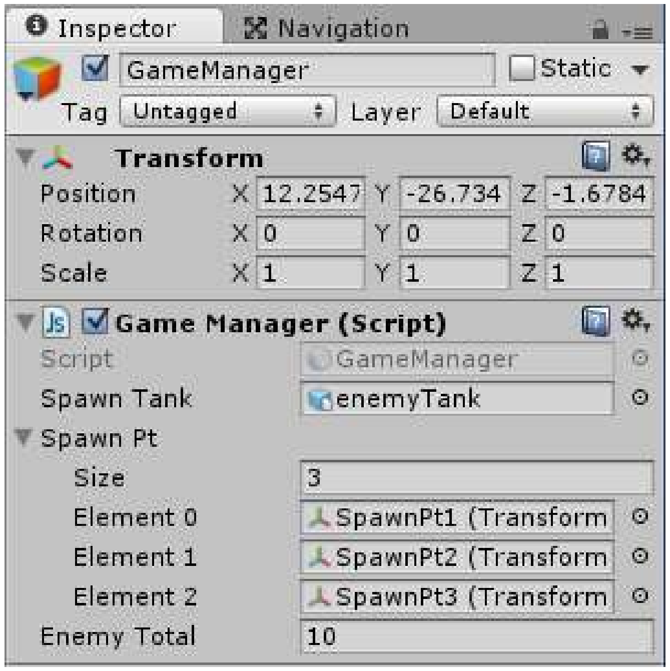
Code Dissection

- SpawnPt is an array holding the spawn points on level
- We use coroutine to spawn your enemies one by one (every 10 seconds) once the level start
- Random.Range is a random number generator which produce a random value between first and second parameter
- Here we try to make it from range 0 to 2

Auto Spawn

- Create 3 empty GameObjects
- Named them SpawnPt(1,2,3) and place them onto different spots in the ground of our level
- Select GameManager, and set the parameter SpawnPt as the created game objects above
- Note once you enter size 3, the editor will open the list to let you add those SpawnPt(1,2,3)
- Initialize the SpawnTank as our enemyTank created in AttackMode





Inspector | Navigation

GameManager — Static

Tag: Untagged — Layer: Default

Transform

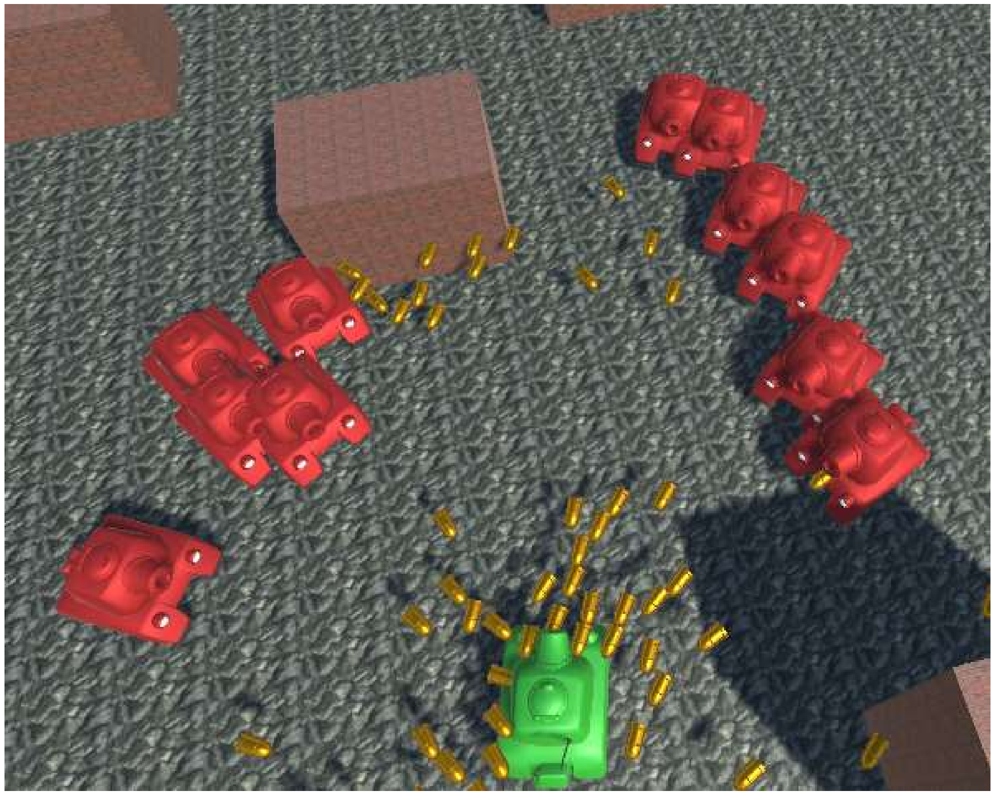
Position	X	12.2547	Y	-26.734	Z	-1.6784
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Game Manager (Script)

Script	GameManager
Spawn Tank	enemyTank
Spawn Pt	
Size	3
Element 0	SpawnPt1 (Transform
Element 1	SpawnPt2 (Transform
Element 2	SpawnPt3 (Transform
Enemy Total	10

Auto Spawn

- Save and run to see how the tanks spawn once 10 seconds

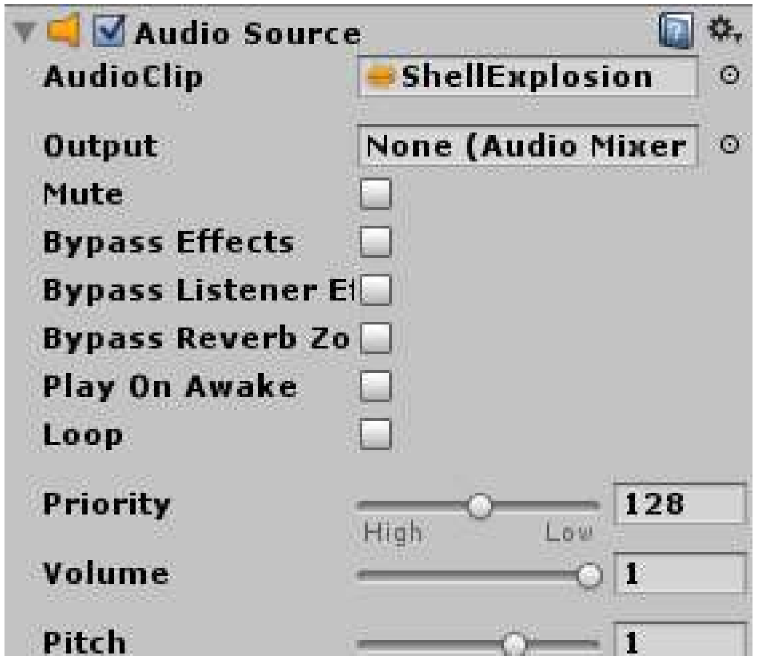


Make Some Noise

- Shell should explode when they contact player or enemy tank
- They should also make sound!
- In Unity, any game object making sound should have an AudioSource component in which we can embed sound file to be played
- You also need to have an AudioListener to listen to all sounds in the game
- Technically the **MainCamera** object that we hook up to the player tank already implemented as AudioListener
- So when we move around the level, we will be listening to the result of all sounds in it

Make Some Noise

- Create an asset script “shell.cs” and made it component of Shell
- Select the shell in editor and choose “Add Component/Audio/Audio Source”
- Set the AudioClip parameter as “shellExplosion” from Assets in our AudioClips folder
- Add the following code to shell.cs



Audio Source	
AudioClip	ShellExplosion
Output	None (Audio Mixer
Mute	☐
Bypass Effects	☐
Bypass Listener Ef	☐
Bypass Reverb Zo	☐
Play On Awake	☐
Loop	☐
Priority (High – Low)	128
Volume	1
Pitch	1

```
private void OnCollisionEnter(Collision otherObj)  {
      GetComponent< AudioSource > ().Play();
      Destroy(gameObject, 0.3f);
  }
```

What to do next?

- Now we have a simple tank game
- But there are many room for improvement
- enemy tanks stay in attack mode
- Enemy fire shells even they are blocked by obstacles
- Player can’t get killed
- Enemy have 360 degree vision capability which is not realistic
- …
- They all can be fixed by scripting!
- Make these changes by thinking & checking Unity documentation