1. Manual

Avoid being destroyed by the enemy cars trying to ram you, for AS LONG AS YOU CAN!!!
- The more enemies you kill the higher the score.
- Kill Enemies one after the other in quick succession (kill streak) and see your score skyrocket.

Controls	
Accelerate	UP ARROW
Turn Left	LEFT ARROW
Turn Right	RIGHT ARROW
Decelerate/Reverse	DOWN ARROW
Shoot Laser Bolts	SPACE
Use Power Up	LEFT ALT
Pause/Unpause	P

In arena 2 the power-up cubes are slowly covered by cubes dropping from the sky. These cube are heavy and pushing them might cost you, even your life!!! Also every time one is destroyed you get some score points.

Every wave cleared gives bonus score points, but can you survive all 8 waves??

Finding it hard to defeat all these enemies coming at you all at once? Collectable power-ups are here to help you. Or are they?

Power-ups	
Health*	+3 health points
Shield	Anything that touches it gets destroyed
Mine	Anything that come in contact with it is destroyed
Mine Exploder	Explodes all mines and cubes that may have dropped from the sky
Booster*	Accelerate faster
Reverse Steering*	Reverses the car steering
Brake Lock*	Causes the car to brake automatically

All power-ups last 4 seconds. Use them or LOSE them.
*used automatically

2. Implemented Features

Game Structures

Main Menu / Pause Menu / End Menu

The main menu is presented just after the splash screen. These are the functions behind the buttons

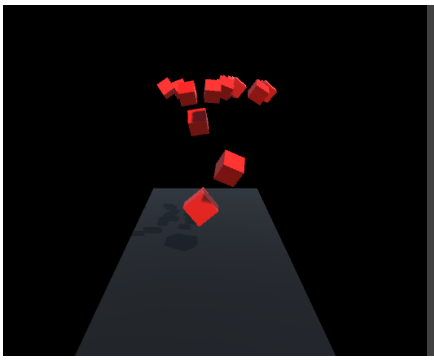
```
public void Level2() {
		SceneManager.LoadScene("Arena 2");
		AudioManager.instance.PlayMusic (AudioManager.instance.mainTrack);
	}
public void Level1() {
		SceneManager.LoadScene("Chase");
		AudioManager.instance.PlayMusic (AudioManager.instance.mainTrack);
	}
public void Quit() {
		Application.Quit ();
	}
```

The code for volume setting was developed but the “Options” button was deactivate for the final submission build, since it had some bugs. I have left the code in though and it can be easily enabled by setting the “Options” Button to active.

```
public void OptionsMenu() {
	mainMenuHolder.SetActive (false);
	optionsMenuHolder.SetActive (true);
}
public void MainMenu() {
	optionsMenuHolder.SetActive (false);
	mainMenuHolder.SetActive (true);
}
public void SetMasterVolume(float value) {
	AudioManager.instance.SetVolume (value, AudioManager.Channel.Master);
}
public void SetMusicVolume(float value) {
	AudioManager.instance.SetVolume (value, AudioManager.Channel.Music);
}
public void SetSfxVolume(float value) {
	AudioManager.instance.SetVolume (value, AudioManager.Channel.Sfx);
}
```

Splash Screen

A small cube falling on a plane and exploding on contact with some added particle effects.



```
public ParticleSystem death;

void OnTriggerEnter(Collider other) {
	Explode ();
}
public void Explode() {
	Destroy (gameObject);
	Destroy (
		Instantiate (death,
		transform.position, Quaternion.Euler
		(new Vector3 (-90,0,0))), 5f);
}
```

Fade In/Out Transitions

Taken form one of the lectures

```
void OnGUI() {
	GUI.color = new Color (GUI.color.r, GUI.color.g, GUI.color.b, alpha);
	GUI.depth = drawDepth;
	Rect dimension = new Rect (0, 0, Screen.width, Screen.height);
	GUI.DrawTexture (dimension, fadeOutTexture);
}

public float BeginFade(int direction){
	fadeDir = direction;
	return 1.0f/fadeSpeed;
}

void OnLevelLoaded(){
	BeginFade (-1);
}
```

Terrain

Created two terrains one for each level. They are both used as backgrounds

---

Gameplay

Collectables / Power-ups

Power-ups	
Health*	+3 health points
Shield	Anything that touches it gets destroyed
Mine	Anything that come in contact with it is destroyed
Mine Exploder	Explodes all mines and cubes that may have dropped from the sky
Booster*	Accelerate faster
Reverse Steering*	Reverses the car steering
Brake Lock*	Causes the car to brake automatically

This script is a component of the Power Up Cube and it controls what happens once the player collides with it.

It randomly picks one of the possible power ups and notifies any subscribed methods. Used to start the time for the power up duration and the UI displays. Also used to set the ammo for the user.

The code below shows how the balance of the randomness was skewed to favour mines above all other power ups.

Gameplay Parameters

An Audio Manager was implemented but don't work as expected for the main volumes so the global volume setting were left out of the game for now. The code used in the menu was provided above. For the code used in the Audio Manager see below.

Art

Audio Manager/Sound Library

The audio manager was designed to control the main volume settings as well as the triggering of background audio and sound effects. The code below is setting the volume based on the channels from the menu and saved the player preferences for the next time.

```
public void SetVolume (float volume, Channel c) {
	switch (c) {
	case Channel.Master:
		masterVol = volume;
		break;
	case Channel.Music:
		musicVol = volume;
		break;
	case Channel.Sfx:
		sfxVol = volume;
		break;
	}

	musicSources [0].volume = musicVol * masterVol;
	musicSources [1].volume = musicVol * masterVol;

	PlayerPrefs.SetFloat ("master volume", masterVol);
	PlayerPrefs.SetFloat ("music volume", musicVol);
	PlayerPrefs.SetFloat ("sfx volume", sfxVol);
	PlayerPrefs.Save ();
}
```

The audio manager has an audio listener as a child that is moved with the player so the sound always comes form the right position.

```
void Update() {
	if (player == null) {
		if (FindObjectOfType<Player> () != null) {
			player = FindObjectOfType<Player> ().transform; } }
	if (player != null) {
		audioListener.position = player.position;
		audioListener.rotation = player.rotation; } }
```

When the scene changes from the menu to an arena the music changes to become more energetic. To do this smoothly I implemented a Transition method that incorporates Lerp to make it smoother.

```
IEnumerator Transition (float fadeDuration) {
	float percent = 0;
	while (percent < 1) {
		percent += Time.deltaTime * 1 / fadeDuration;
		musicSources [1-activeMusicSource].volume =
				Mathf.Lerp (musicVol * masterVol, 0, percent);
		musicSources [activeMusicSource].volume =
				Mathf.Lerp (0, musicVol * masterVol, percent);
		yield return null; } }
```

```
void OnTriggerEnter(Collider c) {
		if (c.CompareTag("Player")) {
			rnd = Random.Range (1, powerups);
			//rnd = Random.Range (1, powerups);
			LivingEntity obj = c.gameObject.GetComponent<LivingEntity> ();
			Player p = c.gameObject.GetComponent<Player> ();
			//debug
			AudioManager.instance.PlaySound ("PowerUp", p.transform.position);
			string power = "No Powerup";

			if (obj != null) {
				switch (rnd) {
				case 1: //mine
					obj.SetAmmo (1);
					power = "Mine";
					break;
				case 2: //shield
					obj.SetAmmo (2);
					power = "Shield";
					break;
				case 3: //slow mo explode all mines
					obj.SetAmmo(3);
					power = "Mine Exploder";
					break;
				case 4: //booster
					p.SetBoosterActive ();
					power = "Booster";
					break;
				case 5: //health
					obj.AddHealth (3);
					power = "Health";
					//show +Health above player
					break;
				case 6: //brake lock
					p.SetBrakerActive ();
					power = "Brake Lock";
					break;
				case 7: //reverse steer
					p.SetReverseSteeringActive ();
					power = "Revesre Steer";
					break;
				case 8: //timer
					obj.SetAmmo (1);
					power = "Mine";
					break;
				case 9: //mine
					obj.SetAmmo (1);
					power = "Mine";
					break;
				case 10: //mine
					obj.SetAmmo (1);
					power = "Mine";
					break;
				case 11: //mine
					obj.SetAmmo (1);
					power = "Mine";
					break;
				case 12: //mine
					obj.SetAmmo (2);
					power = "Shield";
					break;
				}

				if (OnEnpowered != null) { OnEnpowered (power); } }

			GameObject deathParticles = Instantiate(deathEffect,
transform.position, Quaternion.identity) as GameObject;
			Destroy (deathParticles, deathEffect.startLifetime ) ;
			GameObject.Destroy (gameObject);
		}
```

The functions below trigger audio samples that are passed either by name or the actual sample

```
	public void PlaySound(string name, Vector3 pos) {
		PlaySound(soundLib.GetClipFromName (name),pos);
	}

	public void PlaySound(AudioClip clip, Vector3 pos) {
		if (clip != null) {
			AudioSource.PlayClipAtPoint (clip, pos, sfxVol * masterVol);
		}
	}
```

The function above calls this to find the audio sample given the name. To make the game easier to expand in terms of sound design each sound group can have as many sounds needed and a random one is chosen every time.

```
public SoundGroup[] soundGroups;
Dictionary<string, AudioClip[]> groupDict = new Dictionary<string, AudioClip[]>();

public AudioClip GetClipFromName(string name) {
	if (groupDict.ContainsKey (name)) {
		AudioClip[] sounds = groupDict [name];
		return sounds [Random.Range (0, sounds.Length)];
	}
	return null; }
```

Sound Groups	
Death	triggered when a LivingEntity (Player / Enemy) dies
Lazer	triggered when the player is shooting
Explosion	triggered when Mines Explode
Health	triggered when the power up is a Health Boost
New Wave	triggered when all enemies of the current wave are destroyed
Power Up	triggered when colliding with a power up cube
Crash	triggered when the player comes in contact with enemy cars

The above groups include sounds especially designed for this project using samples from different sources. (e.g the death sound is a combination of a hit and a splash sound)

For this game two background tracks were used. A short chilled out loop for the menu and an energetic track for the arenas. Both were composed using Ableton Live and very minimalistic instruments.

Cameras

Used two cameras per scene. One follows the player (Third Person) and another one for the overview shot when the player dies. The cameras that follows the player were animated at the beginning of each the level and travel towards the player until they reach their destination. The player can move as soon as the level starts and while the camera is far away which makes for a cooler effect as the camera is moving towards the player.

Effects

Enemy Death Effect
Cube Destroyed Effect
Power-up Collected Effect
Mine “Bubbling” Effect

Materials

I used mostly monochrome materials and also designed a texture for the floor of the arena.

AI Artefacts

NavMesh Pathfinding

The enemies use the nav mesh to move around and have the player as the target

```
Enumerator UpdatePath () {
	float refreshRate = .25f;
	while (hasTarget) {
		if (currentState == State.Chasing) {
			Vector3 directionToTarget =
				(target.position - transform.position).normalized;
			Vector3 targetPosition =
				target.position - directionToTarget * attackBounds;
			if (!dead) {
				pathfinder.SetDestination(targetPosition);	}	}
		yield return new WaitForSeconds(refreshRate);	}	}
```

When an enemy is very close to the player it attacks by ramming the player and changing colour

```
IEnumerator Attack () {

	currentState = State.Attacking;
	pathfinder.enabled = false;

	skinMaterial.color = Color.red;

	Vector3 originalPosition = transform.position;
	Vector3 attackPosition = target.position;

	float attackSpeed = 3;
	float percent = 0;
	bool hasAppliedDamage = false;

	while (percent<=1) {
		if (percent>=.5 && !hasAppliedDamage) {
			hasAppliedDamage = true;
			targetEntity.TakeDamage(damage);
			AudioManager.instance.PlaySound ("Crash", transform.position); }
		percent += Time.deltaTime* attackSpeed;
		float interpolation = 4*(-Mathf.Pow(percent,2)+percent);
		transform.position =
			Vector3.Lerp(originalPosition, attackPosition, interpolation);
		yield return null; }

	skinMaterial.color = originalColor;
	currentState = State.Chasing;
	pathfinder.enabled = true; }
```