**Asteroids ECS**

by Leandro Wainberg

Unity version: 2020.3.30f1

ECS version: 0.51.1-preview.21

<https://github.com/leandronw/AsteroidsECS>

This game was developed with a hybrid approach, using both Entities/Components/Systems and Monobehaviors.

Main entities in the game are:

* Player: Is represented by a spaceship. Current implementation supports only one Player at a time, although most systems were designed to support many players. Gets destroyed by asteroids, UFOs or UFO’s bullets.
  + Mandatory components:
    - PlayerTag
    - *PlayerInputData*: What player is doing through input.
    - *PlayerAccelerationData*: How fast it accelerates, and max speed.
    - *PlayerRotationData*: How fast it rotates.
    - *WeaponData*: Player can shoot only when this component exists. A default WeaponData is assigned if there isn’t one present.
    - *WrapAroundEdgesComponent*: Needed for appearing on the other side when leaving the screen.
  + Optional components:
    - *PlayerKeyboardComponent*: If present, Player can be controlled with Keyboard. Default component exists in the scene as “Player1KeyboardConfig” (W to thrust, A and D to turn, H to jump to hyperspace, Space to shoot).
    - *WeaponEquipRequest*: Indicates that a new Weapon must be instantiated and linked as a child to Player.
    - *ShieldData*: Player is immune to damage with this component.
    - *ShieldEnableRequest*: Indicates that a new Shield must be instantiated and linked as a child to Player.
    - *CollisionComponent*: Indicates that a collision just happened and was not processed yet.
    - *DestroyedTag*: Marked to be destroyed, shouldn’t be considered part of the world anymore.
* UFO: Can shoot bullets that will kill the Player. Gets destroyed by asteroids and Player’s bullets. Has a very basic AI that tries to avoid asteroids.
  + Mandatory components:
    - *UFOTag*
    - *UFORotatingWeaponData*: Similar to Player’s WeaponData. UFO can shoot only if this component exists, as it contains all the necessary data to do that.
    - *UFOBrainComponent*: Contains necessary parameters for the movement algorithm.
    - *UFOSpeedData*: How fast it moves.
    - *WrapAroundEdgesComponent*
  + Optional components:
    - *CollisionComponent*
    - *DestroyedTag*
* Asteroid: Floats around the screen with random velocity. Gets destroyed by any bullet or collision with Player/UFO. When it’s destroyed, two smaller ones are spawned.
  + Mandatory components:
    - *AsteroidSizeComponent*: Whether it’s big, medium or small.
    - *AsteroidSpeedData*: Speed will be a random value between Min and Max.
    - *AsteroidRotationData*: Angular speed will be random up to Max Speed.
    - *WrapAroundEdgesComponent*
  + Optional components:
    - *NeedsInitTag*: Marks asteroids that still need their velocity set up after spawning.
    - *CollisionComponent*
    - *DestroyedTag*
* Weapon Power-Up: Static object that only collides with Player. When picked, grants a Weapon (permanent).
  + Mandatory components:
    - *WeaponPowerUpTag*
    - *WeaponEquipRequest*: Contains a reference to the prefab that will be instantiated when the weapon is equipped.
    - *WeaponData*: Contains data necessary to shoot.
  + Optional components:
    - *PickedTag*: Whether this power-up has already been picked up.
* Shield Power-Up: Static object that only collides with Player. When picked, grants a Shield (temporary).
  + Mandatory components:
    - *ShieldPowerUpTag*
    - *ShieldData*: Contains a reference to the prefab that will be instantiated when the shield is equipped, and the duration of the effect.
  + Optional components:
    - *PickedTag*

Monobehaviors

# GameManager

Singleton that is responsible for gameplay rules and spawning of main entities:

* Spawns Player when game starts and assigns input controller to it.
* Keeps count of remaining lives on Player’s death, and respawns Player until no more lives are left (Game Over).
* Checks if level was cleared (no more enemies left) and starts a new level, with asteroids and power-ups.
* Spawns UFOs after a certain time interval, which decreases over time.
* Destroys everything from previous game when a new game starts.
* Invokes events for the UI:
  + OnGameStarted()
  + OnGameEnded()
  + OnLivesChanged(int newValue)
  + OnCountdownStarted(float duration)

# GameUI

Listens to events from ECS and GameManager, and shows visual feedback:

* Player lives
* Game Over
* Shield timer
* Countdown

# SfxPlayer

Listens to sound events from ECS and direct calls from Monobehaviors and plays audio clips/loops.

# GameArea

Singleton that keeps track of game world’s edges. Updates itself when screen size changes.

Systems

# ***PlayerKeyboardInputSystem*** (InitializationSystemGroup)

Reads from *PlayerKeyboardComponent* which keys control the player, and writes to *PlayerInputData* the actions the player is doing.

# ***PlayerInputHandlingSystem*** (InitializationSystemGroup)

Reads *PlayerInputData* and acts according to what actions the player is doing:

* Turns player left/right (writes *Rotation*).
* Enables or disables thrust (writes *PhysicsVelocity* and makes child “Thrust” particle system enable/disable)
* Shoots weapon. This spawns a new “Bullet” entity, reading all the necessary info from *WeaponData*. Writes to *WeaponData* by setting ElapsedTimeSinceLastShot to 0.
* Adds *JumpToHyperspaceTag* if player just pressed the hyperspace button.

# ***HyperspaceSystem*** (InitializationSystemGroup)

Make all players that have *JumpToHyperspaceTag* change their position to a random point inside the Game Area and removes *JumpToHypespaceTag*.

# ***CollisionDetectionSystem*** (FixedStepSimulationSystemGroup)

Listens to trigger events from the Physics system, and adds *CollisionData* to collided entities.

# ***CollisionHandlingSystem*** (SimulationSystemGroup)

Reads *CollisionData* from all Entities that collided and acts accordingly:

* Bullets get destroyed.
* UFOs get a *DestroyedTag*.
* Asteroids get a *DestroyedTag*.
* If Player collided with Power-Up, adds a *PickedTag* to Power-Up. If collided with anything else, check if has Shield active, and if not, adds a *DestroyedTag* to Player.

# ***PlayerDestroySystem*** (LateSimulationSystemGroup)

Destroys Player entities with *DestroyedTag*. Spawns a VFX prefab at player’s position.

# ***UFODestroySystem*** (LateSimulationSystemGroup)

Destroys UFO entities with *DestroyedTag*. Spawns a VFX prefab at UFO position.

# ***AsteroidsDestroySystem*** (LateSimulationSystemGroup)

Destroys Asteroid entities with *DestroyedTag*. Spawns a VFX prefab at asteroid’s position. If *AsteroidSizeComponent* is Big or Medium, creates a new *AsteroidsSpawnRequest* for 2 smaller asteroids.

# ***AsteroidsSpawnSystem*** *(*InitializationSystemGroup)

For all *AsteroidsSpawnRequest* instantiates new asteroids and adds a *NeedsInitTag* to them. New asteroid’s *PhisicsVelocity* is set to the value stored in *AsteroidsSpawnRequest*, which is the velocity of the bigger asteroid that was just destroyed.

# ***AsteroidsInitializeSystem*** (SimulationSystemGroup)

Writes the *PhysicsVelocity* of all asteroids with *NeedsInitTag*, using their *AsteroidRotationData*, *AsteroidSpeedData* and previous velocity. *NeedsInitTag* gets removed.

# ***LifetimeSystem*** (SimulationSystemGroup)

Decreases value of *LifetimeComponent* for all entities, destroys them when it reaches 0. Used for bullets and VFX explosions.

# ***WeaponSetDefaultSystem*** (InitializationSystemGroup)

Keeps a reference to the default *WeaponData* and *WeaponEquipRequest*, which are in the scene hierarchy, and assigns them to any Player with no *WeaponData*.

# ***WeaponPickedSystem*** (SimulationSystemGroup)

All picked Weapon Power-Ups (those that have *PickedTag*, *WeaponPowerUpTag*, *WeaponData*, *WeaponEquipRequest* and *CollisionComponent*) get destroyed and their *WeaponData* and *WeaponEquipRequest* are transferred to the Player who picked them, which is referenced in the *CollisionComponent*.

# ***WeaponEquipSystem*** (SimulationSystemGroup)

For Players with *WeaponEquipRequest*, it first destroys any existing Weapon children the Player may have, then instantiates a new Weapon from the Prefab referenced in *WeaponEquipRequest*, and attaches it to the Player as child. *WeaponEquipRequest* gets removed.

# ***ShieldPickedSystem*** (SimulationSystemGroup)

All picked Shield Power-Ups (those that have *PickedTag*, *ShieldPowerUpTag*, *ShieldData* and *CollisionComponent*) get destroyed and their *ShieldData* is transferred to the Player who picked them, which is referenced in the *CollisionComponent*. Also, a new *ShieldEnableRequest* is added to the Player.

# ***ShieldEnableSystem*** (InitializationSystemGroup)

Players with *ShieldEnableRequest* and *ShieldData* get their previously equipped Shield destroyed and a new one is instantiated from the Prefab referenced in *ShieldData*. *ShieldEnableRequest* gets removed.

# ***ShieldDepleteSystem*** (SimulationSystemGroup)

Players with *ShieldData* have their shield TimeRemaining decreased. When it reaches 0 their *ShieldData* is removed and the child Shield is destroyed.

# ***UFOAttackSystem*** (InitializationSystemGroup)

UFOs with *UFORotatingWeaponData* ElapsedTimeSinceLastShot greater than 1 / BulletsPerSecond shoot their weapon spawning EnemyBullets, and set their ElapsedTimeSinceLastShot to 0.

# ***UFOMovementSystem***

When UFOs with *UFOBrainComponent* ElapsedTimeSinceLastChange is greater than MinTimeSinceLastChage, they use the position of nearby asteroids to calculate a new direction, and then calculate a new PhisicsVelocity from UFOSpeedData. The algorithm to calculate the new direction is explained in the “UFO Movement” file.

### *WrapAroundEdgesSystem* (LateSimulationSystemGroup)

Teleports all entities with *WrapAroundEdgesComponent* that are leaving the bounds of the game world to the opposite side.

# ***EventDispatcherSystem*** (LateSimulationSystemGroup)

Consumes event entities and invokes C# delegates that can be listened by Monobehaviors or other systems. Events are:

* OnPlayerDestroyed(float2 position)
* OnUFODestroyed(float2 position)
* OnAsteroidDestroyed(float2 position, AsteroidSize size)
* OnShieldEnabled(float duration)
* OnShieldDepleted()
* OnWeaponEquipped()
* OnHyperspace(float2 fromPosition, float2 toPosition)
* OnSoundPlayed(SoundId sound)
* OnSoundLoopStarted(SoundId sound)
* OnSoundLoopStopped(SoundId sound)