BeatNinja – 3632 Readme

Evan Scott-D’Souza C2030044

# Game Information

## How to play:

* FPS rhythm game.
* Game and level moves to the music
* Dashing [SHIFT] and firing your bow [LMB] will only work if done in time with the beat – use the UI around the reticle to help time your inputs.
* The player can double jump [SPACE].
* Player can dash [SHIFT] into enemies to defeat them, or their bullets. Either will the player upwards and replenish a double jump use.
* Dashing [SHIFT] toward the ground and holding [C] will allow the player to slide
* The player can fire their hitscan bow by holding down LMB for at least the length of one beat and then releasing it on a beat.
* The player can throw grenades [F] that explode (eliminating nearby enemies) after either 4 beats have passed or after being shot by the player’s bow mid-air.
* Grenades have 2 modes – icy and bouncy, swappable with [Q]
* The level is split into sections with requirements blocking progression – eliminating enemies, hitting buttons, or simply reaching the exit in time.
* Falling off the map or being hit by a bullet outside of godmode (toggleable via the main menu)

## External Asset Sources

* Water system - <https://github.com/Unity-Technologies/BoatAttack>
* Music – “Truth to Power”, Official Ghostrunner OST, Daniel Deluxe

# Mark Scheme Applications

Some basic marking points have been combined for brevity (such as having a single and multiple colliders)

# Physics (+30%)

## Newtonian (+5%)

### Rigidbodies

* Used by most moving objects e.g. player

### Correct Impulses

* Jump/Double jump uses ForceMode.Impulse [BasicPlayerMovement.cs]

### Appropriate mass quantities

* Intuitive relative masses
* grenades are 10x lighter than grenades
* Changed in editor (Rigidbody component)

### Physics-Driven Mechanics (e.g. immediate collision response)

(Most mechanics use physics)

* Player movement + double jump mechanic [BasicPlayerMovement.cs]
* Gravity powerup (GravityManager.cs)
* Dash uses rb.velocity (BasicPlayerMovement.cs)
* Grenades use rigidbody physics and Physics package for raycasting and OverlapSphere [Grenade.cs]

## Advanced Physics (+5%)

### Physics Properties changed via scripts

* Velocity property of rb modified many times in BasicPlayerMovement.cs
* Gravity powerup changes Gravity property of standard Unity Physics [GravityManager.cs]

### Mass/Physics is a gameplay mechanic

* As above, the gravity powerup changes Physics properties directly for the scene.
* Player double jump uses physics impulses [BasicPlayerMovement.cs]
* Player dash uses rb.velocity [“”]
* Sliding uses physic material (reduces player friction) [“”]

### Additional forces (e.g. projectile trajectories, gravitational interference)

* Fan objects (FanVectorField.cs) applies additional force on player movement.
* Gravity changes during gravity powerup. [GravityManager.cs]
* Double jump is an additional “AddForce” call used above expected fps mechanics. [BasicPlayerMovement.cs]

### AI uses calculations to determine projectile forces

* ShooterEnemy.cs fires projectiles, and uses the player’s position to determine the correct direction and variable speed to fire the projectile with. This does not directly use AddForce (the bullet directly has its velocity updated each frame) – though the resultant effect is similar.

## Basic Collision Volumes (+5%)

### Multiple Collision Volumes Exist

* Player GO satisfies this.

### Appropriate shape and matches mesh

* Player, ground, enemy and bullet have colliders that match their mesh (others such as grenades intentionally do not).

## Advanced Collision Volumes (+5%)

### Multiple Colliders on a GameObject

* Grenades have a sphere collider (to roll around despite appearing as cubes), and another larger one that acts as a target to be raycasted to by the player using their hitscan bow.

### Scripts enable/disable colliders

* DashManager.cs enables/disables a CapsuleCollider on the player to act as a hitbox on the player to detect when they dash into a nearby target (e.g. a bullet).

### Script (on collider) change collider position

* ButtonDoor.cs uses an animator to change the collider position once shot with the bow.

### Trigger volumes part of player mechanics

* DashTrigger on player used by DashManager.cs to detect when the player dashes into/nearby a valid target.

## Collision Response (+5%)

### Realistic Rigidbody collisions

* Rigidbodies used for player movement and behave as expected

### Use of OnCollisionEnter/Exit

* EnemyBullet.cs, FinishLine.cs, PlrDeath.cs

### Collision Layers used

* Collision matrix in Project Settings used.
* e.g DashManager.cs uses layers to identify which objects are “valid” dash targets such as enemies, bullets and grenades.

## Advanced Collision Response (+5%)

### Multiple Physics Materials

* Apparent in PhysicMaterials Folder.

### Scripts change physics material

* BasicPlayerMovement.cs swaps player’s physic mat to remove friction for sliding.
* Grenades.cs - have an icy or bouncy material (changed when instantiated)

### Trigger volumes trigger gameplay events

* DashManager.cs applies impulse to player on trigger volume collision (core gameplay mechanic)
* Checkpoint.cs

# Graphics (+20%)

## Appropriate Use of Graphical Elements (+10%)

### Multiple Textures appear in game

* Apparent in level geometry (ground, colored blocks)
* Rain splatters are their own texture

### Appropriate lighting

* Apparent in level

### GameObjects move and rotate via script

* BasicPlayerMovement.cs

### Navigable moving 3D camera

* BasicCameraController.cs

## Advanced Graphics (+10%)

### Environment appears to extend infinitely

* Apparent especially at the beginning of the level.

### A body of realistic-looking body of water

* Used to signify the death plane below the level.
* https://github.com/Unity-Technologies/BoatAttack

### Scripted lighting/effects (e.g. weather, day/night cycle)

* DaylightCycle.cs

### Scripts change object appearance

* TargetButton.cs changes color upon raycast from player bow
* Grenade.cs material changes depending on mode (icy/bouncy)

### Geometry that changes over time (e.g plants growing)

* Colored blocks change in time with the music (BeatCube.cs)
* Grenade.cs explosion changes over time (though this is a placeholder)

# Pathfinding (+10%)

### NavMeshAgents Used

* NavShooteEnemy.cs

### NavMeshObstacles Used

* Used by moving enemies, “Center Obstacle” in project hierarchy. Not a core mechanic but demonstrated nonetheless.

### Custom pathfinding code / modifies external lib

* n/a

### AI makes decisions based on pathfinding

* NavShooter (pink mover) enemies will change their state both randomly and based on their pathfinding to the player. If they find themselves too close, they will be forced into their defensive state for 3 seconds and shoot while making distance. [NavShooterEnemy.cs]

# Artificial Intelligence (+20%)

## State Machines (+10%)

### Use of Simple State Machine

* PlayerBow.cs to track bow state (ready to pull or not)
* ShooterEnemy.cs to track when it can shoot and when it is on cooldown

### Boolean or State-Driven State Machines

* BasicPlayerMovement.cs uses Boolean states e.g. isDashing.
* Also above

### Object-Encapsulation to model states

* ILevelStates (LevelStateElim, LevelStateEscape etc)

### Hierarchal state machines / use of external tools to generate state machines

* Pink mover enemies use MoverState to choose between offensive and defensive tactics in NavShooter.cs, though the NavShooter is a child class of ShooteEnemy.cs which also uses a ShootState state to give their shots a cooldown, thus using two state machines on the moving enemies in tandem within their class hierarchy.

### State Machines Triggered by external events / timeouts

* LevelState is triggered by the player reaching new checkpoints.

### Probabilistic / Stochastic transitions

* GravPowerupCube.cs adds random (serialized) variance onto the powerup’s duration on each timer, thus adding randomness to the state transition in GravState (GravityManager.cs)
* NavShooterEnemy.cs also includes variation on how long it will take to transition between Offensive and Defensive strategies

## Advanced AI (+10%)

### Planning Techniques (Real Planners, GOAPs)

* n/a

### Non-cooperative Game Strategies (Min-Max trees, 𝛼-𝛽 pruning)

* n/a

### Basic Reinforcement Learning Techniques

* n/a

# Structuring NPCs (+10%)

### Unorchestrated / react mostly to player or environment

* ShooterEnemy.cs causes enemies to “see” player only when within range and LOS (blockable by environment).

### Coordinated as a group or in tandem to fulfil same goal/task

* Nearby enemies will all attack the player at once in time with the music.

### Orchestrator used to handle multiple and contrasting behaviours incl. random events

* BeatManager.cs allows for any object (including enemy AI) to time code with music given any bpm. Shooter enemies use this to shoot exactly on the beat. Yellow, unmoving shooters will shoot every 4 beats, while pink moving shooters will shoot on every beat.

# Advanced Features (+15%)

### Appropriate Prefabs

* Prefabs folder in project and apparent in hierarchy

### Levels/Menus in distinct scenes

* Scenes folder in project. MainMenu scene and Level scene.

### Evidence of code for limiting expensive computation (e.g. raycasting)

* Raycasts used in PlayerBow.cs for bow hitscan, ShooterEnemy.cs for LOS checks etc.

### Flocking Techniques

* n/a

### Vector Fields

* FanVectorField.cs applies force with ForceMode.Acceleration

### Particle Systems

* Rain and Splash particle systems across level.

### Implementation of Custom AI tools (AverageMinMax etc)

* n/a