alexander.everett@hotmail.co.uk

Beehive Attack
Technical Documentation

Alexander Everett

Project Overview

Platform: Android / Windows Desktop
Development Time: 22 Hours
Unity Version: 2019.4.1f1
Status: In progress

Beehive Attack is a small Android game developed in Unity. This project is more of a development showcase rather than a functional game.

The user controls a 6 legged spider character, whose goal is to steal nectar from the beehive located at the back of the level. The beehive is surrounded by a series of bees working hard to collect nectar from nearby plants and deliver it back to their hive.
The bees have two personalities, worker bees will hide from the player once they get too close to the hive whilst attack bees will intercept the player, retrieving any nectar the player has stolen.

From a development point of view, the exciting part of this application comes from an environment of fully independent objects using a series of controllers and managers to handle multiple state machines to assign appropriate objectives depending on world events.
This application was developed with optimisation in mind with careful consideration when it comes to utilising the physics engine and the navigation agents.

Environment / world assets provided by the Unity Store.

alexander.everett@hotmail.co.uk

World Object Rules / Specifications

Spider

Description:
The spider is controlled by the user. Its purpose is to get close to the hive to steal the nectar and escape the presence of any bees whilst in the “detection zone” (The visible field around the hive).

Rules / Specifications:

- Path finding navigation using mouse click (Windows) and touch inputs (Android).
- Joystick control for camera orbit.
- Receive nectar from the hive once within range, providing the hive has nectar available.

Bee

Description:
The bees movement is handled by AI using navigation mesh agents. Once they receive a new objective they calculate the quickest path and head to the assigned location. The bees currently have two personalities; worker bee and attack bee, with a third personality in development. The bees will always have an objective.

The bees behaviour is determined by their state in the world such as working, attacking, defending etc and also by the state of their nectar controller. The bees themselves are mainly focused on their work objectives which are obtaining nectar from various flowers around the scene.

Rules / Specifications:

- Path finding navigation depending on their work objective and events in the world.
- Bees communicate with the game manager (hive manager) to receive a list of work objectives to retrieve nectar.
- Bees will check their next work objective to ensure there are no more than two bees currently working at that objective, otherwise they will skip to the next work objective.
- Bees will spend a limited amount of time at each work objective to help prevent depleting the work objective entirely of nectar.
- The bees will return to the hive once they cannot carry anymore nectar.
- Once the bees have delivered all nectar to their hive, they will resume their work objectives.
- If an intruder is detected near the hive, defense bees will move to the work objective that is closest between the current work objective and their previous work objective
- If an intruder is detected near the hive, attack bees will move directly towards the player. Once within range they will start removing nectar from the player.
- Once an intruder is clear from the hive, bees will resume their last objective either a work objective or returning to the hive. .

Flowers

Description:
Flowers are the work objectives used by bees to obtain nectar.

Rules / Specifications:

- Once they are intercepted by bees they will decrease their nectar value. The rate in which the flower depletes is dependent on how many bees are collecting from it.
- Flowers will regenerate their nectar over time at a constant rate providing there are no bees collecting from them.

Beehive (Hub)

Description:
The hub is the center of attention. It is the main objective for the player to steal the nectar collected by the bees once they drop off their supply.

Rules / Specifications:

- The hub will naturally decrease in nectar at a constant rate over time providing more work for the bees.
- The hub will send out an alert to all bees once the player gets within its detection zone causing the bees to change their behaviour.
- The hub provides feedback to the user regarding its alert state by changing the colour of its shield.
- Once the player is within range of the hive itself, the hub will start decreasing its nectar supply for the duration of the player's presence or until the player is intercepted by attack bees.

Overview of Scripts and Techniques

HiveManager.cs

Class Type: Singleton **Derived From**: Monobehaviour
Description:
This manager class is only used to look over the work objectives used by the bees. By utilising this manager in any level, developers / designers are free to add any objectives to the levels without the need for any code changes.

Techniques:
- Singleton class

LevelManager.cs

Class Type: Singleton **Derived From**: Monobehaviour
Description:
This manager class is used to handle transitioning between maps. Using the OnLoadNewLevel method, the application will travel to the designated transition map whilst the desired map loads in the background using asynchronous operations..

Techniques:
- Singleton class
- C# Events
- Coroutines
- Async Operations

DebugManager.cs

Class Type: Singleton **Derived From**: Monobehaviour
Description:
Debug managers are a great way to allow developers to efficiently test their code. As this application was built in a short period of time, there wasn't much added here.
When working with stand alone builds on devices, it can be troublesome accessing runtime data whilst debugging. This class provides an output window in the application to read such information.

Note: Set the "EnableOnScreenLog" to true in the DebugMananger located in the _GM game objective inside the BeehiveAttack level to view the on screen output log.

Techniques:
- Singleton class

ClickToMove.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
This class is used to contain the movement behaviour of the player. The code uses macros to determine which platform the application is running and executes the appropriate code. For instance deploying to mobile will only execute the touch navigation calculations whilst running on Windows will execute both (required for Unity Remote emulation)..

This code sequence also filters touch inputs intercepting game objects and UI objects. This is used to prevent moving the player if the user is interacting with UI elements.

Techniques:
- Macros
- Unity Events
- Dynamic Loading of Assets
- Raycast Using Component
- Mobile Touch Input Detection

JoystickController.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
The Joystick controller is used for the visual handling of the on screen joystick. Objects which want to listen to the joystick can bind themselves to the JoystickOutput event.

Techniques:
- Using interfaces IDragHandler and IEndDragHandler
- Custom Unity Event Classes

MainMenuController.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
Very small class to handle the UI button events to start or exit the game.

Techniques:
- Basic Unity C# Only.

OrbitController.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
Basic little script to orbit the camera around a specific object with restrictions.

Techniques:
- Basic Unity C# Only

SpiderController.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
The SpiderController class is used to prepare all the objects required for the spiders behaviour during runtime including the detection when intersecting with the hive object itself.

Techniques:
- Getters / Setters
- Unity Events
- Unity Event Property Bindings
- Coroutines
- Initialisation Sequencing
- Collision Handling

BeeController.cs

Class Type: Unity Class **Derived From**: Monobehaviour
Description:
The Bee Controller handles receiving and validating the work objectives for the bee. It also acts as a public interface for the bees behaviour controller.

Techniques:
- Getters / Setters
- Unity Events
- Custom Unity Event Classes
- Coroutines
- Initialisation Sequencing

BeeBehviourController.cs

Class Type: C# Class **Derived From**: None
Description:
The bee's behaviour controller works with the bee controller and nectar controller to handle the two different state systems which influence the bees focus on objectives.

Techniques:
- Getters / Setters
- Event Bindings

HiveController.cs

Class Type: Unity Class **Derived From**: MonoBehaviour
Description:
The class used to validate what is inside the beehives (hubs) detection zone

Techniques:
- Collision Handling

NectarController.cs

Class Type: Unity Class **Derived From**: MonoBehaviour
Description:
This is the controller which handles the distribution and release of nectar throughout. Objects are assigned a scriptable object called "Nectar Profile" which stores the properties which influence the behaviour of a specific nectar controller.

Techniques:
- Scriptable Object Instance Handling
- Coroutines
- Unity Events

NectarControllerProfile.cs

Class Type: Scriptable Object **Derived From**: ScriptableObject

Description:
The nectar controller profile is essentially the brains behind the nectar controller class. Each object is assigned one and their different configurations determine how the nectar controller class responds to events in the world. The scriptable objects are located in:
“Assets/Resources/ScriptableObjects”

The class uses structures called “Senders” and “Receives”. These determine how a particular object handles the distribution of nectar, whether it sends nectar to another object, i.e. the bee to the hive, or receives nectar i.e. the bee to a flower.

Most of the functionality is handled in the getters and setters of properties. If for instance it has been depleted of nectar, it will notify the owning controller of this so it can pass the response to the world.

Another example is how the “numberOfReceives” / “numberOfSenders” work. Objects such as the flowers can have multiple bees collecting nectar from them, so they use these properties to handle their own states and communicate these updates with the owning nectar controller.

Techniques:
- Getters / Setters
- Virtual Methods (Inheritance)
- Event Handling

SpiderNectarControllerProfile.cs

Class Type: Scriptable Object **Derived From**: NectarControllerProfile

Description:
Derived from the Nectar Profile object, the spider profile uses its “numberOfSenders” property to invoke the events which notify when the player is being attacked or not.

Techniques:
- Getters / Setters
- Override Methods (Inheritance)

InstanceGenerator.cs

Class Type: Unity Class **Derived From**: MonoBehaviour
Description:
A support class used to spawn instances of an object through an event system. The object will keep a recorded list of all objects spawned with methods to manage them.

Techniques:
- Getters / Setters
- Unity Event System
- Virtual Methods (Inheritance)
- Method Overloading

BeeGenerator.cs

Class Type: Unity Class **Derived From**: InstanceGenerator
Description:
A class to spawn and manage instances of the Bee prefab specifically.

Techniques:
- Method Overriding (Inheritance)

UIBillboard.cs

Class Type: Unity Class **Derived From**: MonoBehaviour

Description:
A simple class used to rotate an object to face the camera

Techniques:
- Basic Unity C# Only

Enums.cs

Class Type: C# Class **Derived From**: None
Description:
A data class used to store enumerators

Techniques:
- Namespaces
- Enumerators

Structs.cs

Class Type: C# Class **Derived From**: None
Description:
A data class used to store structures

Techniques:
- Namespaces
- Structures

EventClasses.cs

Class Type: C# Class **Derived From**: None
Description:
A data class used to custom event classes used to pass arguments

Techniques:
- Namespaces
- Event Classes

Development Process

Project Intentions

Beehive Attack will not be winning any Baftas, although this was intended to be more of a development showpiece than a playable game.

The objective was to build an application with multiple state systems which naturally handles every scenario keeping all systems as stable as possible. Everything was built modular so each object looks after itself and should any issues arise, it will not affect the rest of the game. Since everything is modular, designing future levels will not be an issue as the code takes care of handling situations based on the environment. This has been tested with 15 work objectives and 60 bees, without any effect in performance.

Multiple input systems using touch and mouse control was also part of the plan from the start. Working with hardware with limited capabilities forces a need to optimise code and work in a defensive manner.

This was also designed as a team project, so multiple people can work on various mechanics without there being issues with code merges or conflicts. This is the main reason for all data objects such as Enums, Stucts and Event Classes are kept separate.

Final Thoughts

There is no doubt this project could be simplified. Replacing the bees with scriptable objects would significantly decrease the need for some of the event systems. However, as this is a development piece I wanted to show examples of uses different coding approaches such as using external C# classes to compartmentalise specific behaviour.

When planning this project, I believed there would be a good need for more OOP features such as abstract / overloading / overriding / polymorphism etc. Although once the code structure had been designed, I found there wasn't a greater need as first believed. I have tried to implement at least some examples of OOP through inheritance.

I really enjoyed putting this little game together and I do have ambitions to expand on it such as:

1) Convert the Nectar Controller into a base class and separate the bee and spider behaviour into child classes.
2) Add a Queen Bee as a boss which naturally spawns when the nectar level on the hive gets below a certain limit.
3) Add a level completed screen with a couple more levels
4) A score system based on time.

Finally, thank you for reviewing this project. If you have any further questions regarding my approaches or chosen practices for this application, please feel free to drop me an email.