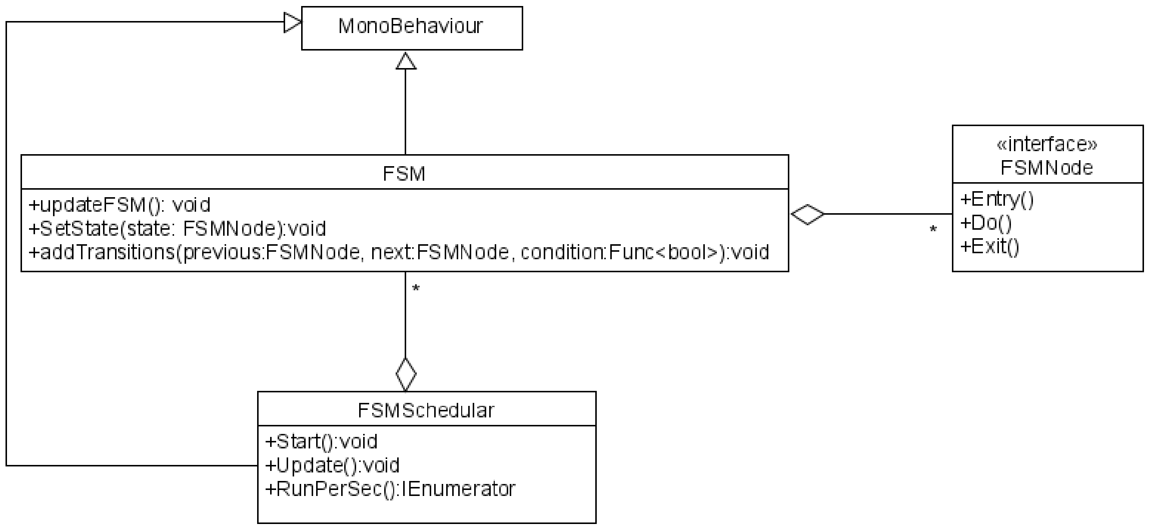
Name: Grace Zang
Student Number: 20065628
Assignment 2 Part 4 Report

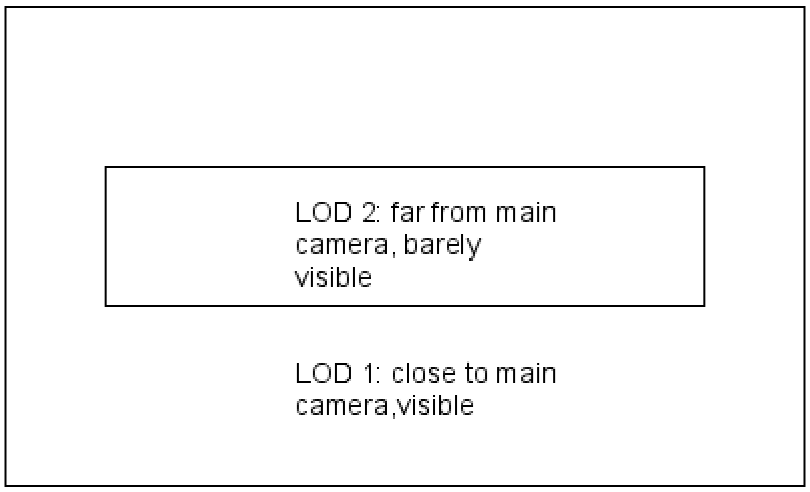
1.



Note: only public attributes/operations/classes are listed

1. FSM Scheduler class runs two instances of FSMs, check and assign priority level to instances of FSM in Start(), updating one FSM with higher priority in Update(), and one FSM with lower priority in IEnumerator.
2. FSMNode uses an interface to achieve full abstraction. It enforces classes inherited from this interface to implement all methods it has: which is Entry(), Do(), Exit().
3. FSM class uses instances of FSMNode , calls functions of FSMNode to set up states and transition logics. updateFSM() sets state to next if there is a transition, else keep updating current state. SetStates method is made public so the initial state can be customized outside of the framework. addTransitions method uses type Func<bool> as condition, making evaluation of multiple booleans simpler.

2.
Since the camera is static and the scene is third-person (no player character) for this assignment, we can assign priority to AI objects based on their position regarding the main camera. Diagram below demonstrate this:


LOD 2: far from main camera, barely visible
LOD 1: close to main camera,visible

LOD3: not in area of main camera, not visible

When a villager is in the LOD 1 area, it has the highest priority and it gets to update every frame to allow smooth animation. While the villager enters LOD 2, LOD 3 area, the update frequency decreases and the animation will seem jumpy.
In our assignment, tree1 is in the LOD 1 area, village is in the LOD2 area, tree3 is in the LOD3 area. The FSM scheduler will get the state the villager is in, and if it gets the villager is now in the village, then it will run that villager in a coroutine that updates every 0.1 seconds. Similarly, if the FSM scheduler gets the villager is picking tree3, then it will schedule that villager to update every second.

3.
I think it is difficult for graphical editors to have proper fsm behaviour since there is only limited space to fit information in a graph. As we know, a finite state machine has multiple condition checks for even one transition, and this is difficult to specify clearly without compromising the neat structure of a graph. E.g. CanCarryMore transition is associated with many booleans checks.

```
addTrans(pickingFruits, findingTree, CanCarryMore());

Func<bool> CanCarryMore() => () => TargetFruitTree != null &&
                                   tree.runOut() &&
                                   _inHand < _carryCapacity &&
                                   anim.GetCurrentAnimatorStateInfo(0).IsName("idle");
```

Also, connections are the key to a finite state machine, and it is hard to represent inheritance in a graphical editor. In other words, it is easy to write Enter(), Do(), Exit() in a graphical editor, but it is hard to connect that to FSMNode and FSM and all states.
Thirdly, the information about how to carry out actions can only be coded. In the graph editor, we can only specify “walk avatar to target at given speed”, but without getting components, using vector3, the avatar can not carry out such action.
Last but not least, it will be difficult to debug in a graphical editor.