Enemies (Developed by Kevin)

Many of the enemy’s behavior will be controlled by a manager script, such as checking for line of sight, checking if a shot hits, and possibly controlling how the tentatively proposed Hunter enemy would move around the level. Some debug information may be displayed by the manager script.

**Enemy Manager Script**

* **Attributes**:
  + Configuration:
    - (Bool[]) Show Debug Information: Depending on which positions in the array are true, activate certain debug methods by default. Dictionary might be better, will do some research later.
  + State:
    - (List<Enemy>) List of enemies in the level. Iterated through in loops. Cleared (if it had anything) and then populated when entering a level.
    - (GameObject) Player. Reference to player GameObject which will be passed into each enemy’s LineOfSight(), Hear(), and/or Attack() method in loops.
* **Methods**:
  + Normal:
    - public void CheckLevel(): Removes all current enemies (if any) from enemy list and adds all active enemies to enemy list. **This method should be called every time the level changes by the Scene Manager script.**
    - private void MainLoop(): Method which contains a loop that calls every enemy’s MainLoop() method. May also contain debugging functionality that can be toggled.
    - public void PauseAll(): Pauses physics and behavior of all enemies in the level by calling each enemy’s Pause() method in a loop.
    - public void UnpauseAll(): Exactly what you’d think.
  + Debug:
    - public void ShowAllDebug(): Activate every enemy’s ShowAll() method, which will display their vision area, hearing area, idle search duration, current attack windup timer, current attack winddown timer, attack details (e.g. raycasts), current state, previous state, and direction facing.
    - public void Show--**X**--Debug(): Activate every enemy’s --**X**-- debug show methods. (There will be many of this type of method, but rather than clog up the already very dense code design doc, I will just write this.)
    - public void HideDebug(): Hide every enemy’s debug information.
    - public void ToggleEyes(): Makes all enemies continue to perform normally except they can’t “see” the player anymore. If called again, re-enables enemy “sight”.
    - public void ToggleEars(): Makes all enemies continue to perform normally except they can’t “hear” the player anymore. If called again, re-enables enemy “hearing”.
    - public void ToggleAI(): Makes all enemies stop doing anything. If called again, re-enables enemy behavior.
    - public void ForceAttackAll(): All enemies in the level will try to attack the player at once, regardless of line of sight. This method could be used to debug cover collision.
    - public void KillAll(): Kill all living enemies in level by calling their Die() method.
    - public void ResAll(): Resurrect all dead enemies in level.

**Enemy Base Class**

* **States**:
  + Patrolling
  + Attack Windup
  + Attacking
  + Attack Winddown
  + Actively Searching
  + Idly Searching
  + (?) Being Mauled
  + Dead
* **Attributes**:
  + Configuration Variables:
    - (Float) Patrol Move Speed
    - (?) (Float) Patrol Turn Delay Duration
    - (?) (Float) Player Check Delay Duration (performance reasons maybe)
    - (Float) Vision Length
    - (Float) Vision Arc
    - (Float) Hearing Range
    - (Float) Active Search Move Speed
    - (Float) Idle Search Duration
    - (Float) Attack Windup Duration
    - (Float) Attack Winddown Duration
    - (?) (Float) Bleed Rate
    - (?) (Float) Bleed Distance
  + State Variable:
    - (Float) Attack Windup Timer
    - (Float) Attack Winddown Timer
    - (?) (Float) Patrol Turn Delay Timer
    - (?) (Float) Player Check Delay Timer (performance reasons maybe)
    - (Float) Idly Searching Timer
    - (Bool) Direction Facing
    - (Enum) Current State
    - (Bool) CanAct (technically this variable, CanAct, and CanHear are debug variables, but they directly affect how the enemy behaves, so I think it is more appropriate to put them here)
    - (Bool) CanSee (see above comment)
    - (Bool) CanHear (see above comment)
  + Holder Variables:
    - (Vector2) Direction Enemy To Player
  + Debug Variables:
    - (Enum) Previous State
    - (Bool) Show All Debug Info
    - (Bool) Show Vision “Cone” [draw three debug lines]
    - (Bool) Show Hearing “Circle” [draw semi-transparent circle sprite]
    - (Bool) Show Current Idle Search Time [draw text near enemy]
    - (Bool) Show Attack Windup Timer [draw text near enemy]
    - (Bool) Show Attack Winddown Timer [draw text near enemy]
    - (Bool) Show Attack Details [draw debug line w/ different colors based on collision]
    - (Bool) Show Previous State [draw text near enemy]
    - (Bool) Show Current State [draw text near enemy]
    - (Bool) Show Direction Facing [show arrow sprite flipped one way or the other]
* **Methods**:
  + Normal:
    - protected abstract void MainLoop(): Perform all the behavior based on current state while checking for state changes. Whenever state is changed, previous state is saved in a variable. May contain debug functionality.
    - private void Die(): Change state to “Dead” immediately
    - private void Res(): Change state to “Patrolling” if previous state was “Dead”.
    - public void LineOfSight(GameObject player): (?) Every couple of seconds (/?), create a raycast between approximately where head is on sprite and the player’s position\*. If the raycast returns the player, change state to “Attack Windup” state.
      * **This method is called by the enemy manager script when the enemy’s state is “Patrolling”, “Actively Searching”, or “Idly Searching”.**
    - public void Hear(Vector3 playerPosition): (?) Every couple of seconds (/?), do simple distance check between position of enemy and position of player. If distance is less than Hearing Range, change direction to face player and change state to “Actively Searching” state.
      * **This method is called by the enemy manager script when the enemy’s state is “Patrolling”, “Actively Searching”, or “Idly Searching”.**
    - public void AttackWindup(): Behave differently based on type of enemy. For human guards, immediately start the Attack Windup Timer. For wolf guards, change state to “Actively Searching”. (In the MainLoop() method for human guards, when the timer finishes change state to “Attacking”. For wolf guards, start Attack Windup Timer when within leaping range of player.)
    - public void Attack(GameObject player): Depending on the type of enemy, perform an attack. For guards with guns, create raycast between approximately where their gun is on their sprite and the player’s position\* and begin the Attack Winddown Timer. If that raycast hit the player, presumably call the player’s Die() method. For wolf guard, leap forward, as well as doing some timer stuff that will be detailed in the wolf section, but basically check Collision2D and if player is collided with, call the player’s Die() method. After the Attack Winddown Timer finishes, set state to “Patrolling”.
      * **This method is called by the enemy manager script when the enemy’s state is “Attacking”.**
    - \*Technically four raycasts will be checked: one for a point representing each corner of the player’s collider. If any of those raycasts return the player, proceed. I do it that way because if I just checked the player’s normal transform (their midpoint), the player could possibly be half below a platform and seem to be visible to the enemy, but in reality would not be seen/hit by the enemy.
  + Debug:
    - public void ShowAllDebug(): Show all the debug information about this enemy.
    - public void Show--**X**--Debug(): Show **X** specific debug information about this enemy.
    - public void HideDebug(): Hide all debug information about this enemy.
    - public void ToggleAI(): Toggles the variable CanAct true or false, which is checked at the very beginning of MainLoop().
    - public void ToggleEyes(): Toggle the variable CanSee true or false, which is checked in the MainLoop() before calling LineOfSight().
    - public void ToggleEars(): Toggles the variable CanHear true or false, which is checked in the MainLoop() before calling Hear().
    - public void ForceAttack(): This enemy will try to attack the player, regardless of line of sight. This method could be used to debug cover collision. May look very dumb with wolf enemies.
    - public void Res(): If this enemy’s state is set to “Dead”, set it to “Patrolling” state.

Basic Human Guard

* Overrides and Unique Methods

Basic Wolf Guard

Hunter

Player (Wolf Form)

Since there isn’t a section for player yet, I’m going to start one to doc my animation states

**Animation States**

Idle

Run

Crouch (Charging for jump)

Lunge

Dead

Transforming

Engaged (Aggressor)

Engaged (Prone)

Walking / sneaking ?