actieplan

willem seynaeve

March 3, 2016

Contents

1 Position, name, attributes, HP & SP, orientation

Essentially just an exercise in getters, setters, and checkers in total, nominal, and defensive programming. It is doable (though still a lot of work, don't get me wrong).

2 advanceTime **METHOD**

```
public void advanceTime(double deltaTime)
```

Unit is a finite state machine.
There are transition flags. In descending order of importance, they are:

- shouldRest
- shouldWork
- shouldAttack

Whether the Unit should move is determined by the non-emptiness of at least one of: immediateTarget and path.
There is also the sprinting flag: sprinting.
There are 7 states:

- NOTHING
- MOVING
- RESTING$_{\text{INIT}}$
- RESTING$_{\text{HP}}$
- RESTING$_{\text{STAMINA}}$
- WORKING
- ATTACKING

Don't forget to check if deltaTime was maximally 0.2 seconds!

2.1 When in NOTHING

If there is an immediateTarget, go to MOVING. Set all flags to low.

ElseIf there is a path, put the first element in immediateTarget, remove that element from the path, and go to MOVING. Set all flags to low.

ElseIf shouldRest is high, do transitionToRestingInit.

ElseIf shouldWork is high, do transitionToWorking.

ElseIf shouldAttack is high, do transitionToAttacking.

ElseIf defaultBehaviour is high, choose a random option from

- Ordering yourself to move to a random position in the game
- Ordering yourself to work by setting the shouldWork flag high
- Ordering yourself to rest by setting the shouldRest flag high

(If defaultBehaviour is low, do nothing and remain in this state.)

2.2 When in MOVING

If reachedImmediateTarget() && !path.isEmpty(), set Unit position to center of immediateTarget. Place next element of path in the immediateTarget.

ElseIf reachedImmediateTarget() && path.isEmpty(), set Unit position to center of immediateTarget. Do transitionToNothing.

ElseIf !reachedImmediateTarget(), update position as explained in pdf. The x' from the pdf is the x value of immediateTarget; the x is the x value of position; the v can be calculated from the attributes, the sprinting flag, and the difference between the current position and the immediateTarget; the new position can be calculated from v and the given deltaTime. Don't forget to lower the stamina by 1 if sprintingStaminaDecreaseCountdown reaches 0. In that case, reset the countdown to its top value (0.1 s) minus the negative overshoot. Also, if the stamina itself reaches zero, set the sprinting flag to low.

2.3 When in RESTING$_{\textbf{INIT}}$

If restingInitialCountdown > 0, decrease it with deltaTime.

Else, do transitionToRestingHP.

2.4 When in RESTING$_{\textbf{HP}}$

If shouldWork is high, do transitionToWorking.

ElseIf shouldAttack is high, do transitionToAttacking.

ElseIf HP is full, do transitionToRestingStamina.

Else(If HP is not full), decrease restingHPCountdown by deltaTime. If it reaches 0 this way, add 1 HP to the total and reset the countdown to its top value, minus the negative overshoot.

2.5 When in RESTING$_{\textbf{STAMINA}}$

If shouldWork is high, do transitionToWorking.

ElseIf shouldAttack is high, do transitionToAttacking.

ElseIf HP is full, do transitionToRestingHP.

ElseIf stamina is full, do transitionToNothing.

Else(If stamina is not full), decrease restingStaminaCountdown by deltaAtime. If it reaches 0 this way, add 1 stamina to the total and reset the countdown to its top value, minus the negative overshoot.

2.6 When in WORKING

If shouldRest is high, do transitionToRestingInit.

ElseIf shouldAttack is high, do transitionToAttacking.

ElseIf workingCountdown $<= 0$, do transitionToNothing.

Else(If workingCountdown > 0), decrease workingCountdown by deltaTime.

2.7 When in ATTACKING

If attackingCountdown is > 0, decrease attackingCountdown by deltaTime.

ElseIf inRangeForAttack(victim), call the defend method on the victim, and pass this Unit to give the needed information. Do transitionToNothing.

Else(If !inRangeForAttack(victim)), do transitionToNothing.

3 Movement

3.1 moveToAdjacent **METHOD**

`public void moveToAdjacent(CubeLocation destination)`

Only if in allowing state: NOTHING, RESTING$_{\text{HP}}$, RESTING$_{\text{STAMINA}}$, WORKING: immediateTarget is set to destination.

3.2 moveTo **METHOD**

`public void moveTo(CubeLocation destination)`

Only if in allowing state: NOTHING, RESTING$_{\text{HP}}$, RESTING$_{\text{STAMINA}}$, WORKING: path is set to an ArrayList of consecutive (opeenvolgende) CubeLocations to be followed cube by cube.

4 Combat

4.1 attack **METHOD**

`public void attack(Unit victim)`

Only if in allowing state: NOTHING, RESTING$_{\text{HP}}$, RESTING$_{\text{STAMINA}}$, WORKING: Set the shouldAttack flag high. There is an internal victim variable, which is used later on, when actually attacking. Set the victim variable to passed victim.

4.2 defend **METHOD**

`public void defend(Unit attacker)`

An instantaneous response to the attack. Everything is handled immediately: dodging, blocking, damage taking, teleportation. The state of the defendant is set to NOTHING using transitionToNothing, and this method is the only one to break the FSM model. attacker is used to get information about damage done.

5 rest METHOD

```
public void rest()
```

Only if in allowing state: NOTHING, WORKING: The shouldRest flag is set to high.

6 work METHOD

```
public void work()
```

Only if in allowing state: NOTHING, RESTING$_{\mathrm{HP}}$, RESTING$_{\mathrm{STAMINA}}$: The shouldWork flag is set to high.

7 Extra stuff

7.1 Helper classes

7.2 Helper variables

7.2.1 previousPosition

```
private double[] previousPosition
```

Holds the previous position of the Unit. Very important to determine whether it has reached its destination.

7.2.2 immediateTarget

```
private double[] immediateTarget
```

The place that the Unit is currently going.

7.2.3 path

```
private List<double[]> path
```

A list of positions that the Unit should walk towards, in correct order.

7.2.4 sprintingStaminaDecreaseCountdown

`private float sprintingStaminaDecreaseCountdown`

The time it will take before the next whole point of stamina is subtracted from the Unit's stamina gauge.

7.2.5 restingInitialCountdown

`private float restingInitialCountdown`

The time it will take before the initial resting period is over, and the Unit transitions to a real RESTING_... state.

7.2.6 restingHPCountdown

`private float restingHPCountdown`

The time it will take before the next whole point of HP is restored by resting.

7.2.7 restingStaminaCountdown

`private float restingStaminaCountdown`

The time it will take before the next whole point of stamina is restored by resting.

7.2.8 workingCountdown

`private float workingCountdown`

The time it will take before the work is done.

7.2.9 attackingCountdown

`private float attackingCountdown`

The time it will take before the attack is actually carried out.

7.2.10 victim

`private Unit victim`

The unit that will be attacked once the attackingCountdown is done.

7.3 Helper methods

7.3.1 transitionToNothing

```
private void transitionToNothing()
```

Set state to NOTHING. Set all flags to low.

7.3.2 transitionToRestingInit

```
private void transitionToRestingInit()
```

Set state to RESTING$_{\text{INIT}}$. Set restingInitialCountdown to the time it would take to restore 1 HP (see pdf or preliminary). Set all flags to low.

7.3.3 transitionToRestingHP

```
private void transitionToRestingHP()
```

Set state to RESTING$_{\text{HP}}$. Set restingHPCountdown to the time it will take to restore 1 HP (see pdf or preliminary). Set all flags to low.

7.3.4 transitionToRestingStamina

```
private void transitionToRestingStamina()
```

Set state to RESTING$_{\text{STAMINA}}$. Set restingStaminaCountdown to the time it will take to restore 1 stamina (see pdf or preliminary). Set all flags to low.

7.3.5 transitionToWorking

```
private void transitionToWorking()
```

Set state to WORKING. Set workingCountdown to the time it takes to complete the work (see pdf or preliminary.org). Set all flags to low.

7.3.6 transitionToAttacking

```
private void transitionToAttacking()
```

Set state to ATTACKING. Set attackingCountdown to the time it takes until you can attack (see pdf or preliminary.org). Set all flags to low.

7.3.7 reachedImmediateTarget

```
private boolean reachedImmediateTarget()
```

Checks whether the Unit has reached the immediateTarget, by overshooting it by some distance.

```
if (between(x_immediateTarget, x_prev, x_cur) ||
     between(y_immediateTarget, y_prev, y_cur) ||
     between(z_immediateTarget, z_prev, z_cur)
     )
     return true;
```

7.3.8 inRangeForAttack

```
private boolean inRangeForAttack(Unit victim)
```

Checks whether the victim is in range for the attack (in the same or an adjacent cube).