Super 6.009 Adventure (Part 2)

Submission to website (part 2): Wed, May 17, 3PM (end of last class period) **

Checkoff by LA/TA (part 2): Thu, May 18, 10PM (last day of classes) **

** Note that these final deadlines, imposed by Institute rules, cannot be extended via the use of late days, sorry.

This lab assumes you have Python 3.5 or later installed on your machine. Please use the Chrome web browser.

The usual collaboration policy applies. Showing another student the web UI running your code is OK; showing them your code is not OK.

Concrete tasks to complete are marked with ⇨. There are 10 coding points for this lab awarded for good organization and class hierarchy, minimizing the amount of repeated code, useful comments.

Schedule

This lab is about twice as long as the first seven labs; accordingly, it is distributed over two weeks:

- In week 1, we'll implement basic collision detection and collision resolution, as well as a few simple items and basic motion.
- In week 2 (this lab), we'll add faster collision detection, more blob types, nice-looking jumps, items, and foes.

This zip contains tests for both the first week and second week to ensure that additions you make this week don't break last week's work! But only tests 23 -- 39 will be run when you submit online.

Week 2

You should start by copying over your lab.py from Week 1 -- that code will have to be in good working order to complete this week's tasks.

Technical details

Several of the following parts use notions of "vertical" and "horizontal" collisions. A collision is vertical if the corresponding minimal translation vector is vertical, and horizontal otherwise.

Jumps [tests 23 -- 26]

Our initial implementation of jumps was rather simplistic. Let's fix it:

⇨ Change the event handling for "up" so that pressing "up" sets the player's vertical speed to `Constants.PLAYER_JUMP_SPEED` for the next `Constants.JUMP_DURATION` rounds.

This requires a bit of clarification. If the player starts at `y = 0`, and the user jumps at time `t = 0`, then here is how `y` evolves, assuming that `PLAYER_JUMP_SPEED` is `62`, `JUMP_DURATION` is `3`, `GRAVITY` is `-9`, and `MAX_DOWNWARDS_SPEED = 48`

```
Step n: vy is … and y is…
0       0       0
1       53      53   # Initial upwards accel.; speed is JUMP_SPEED - GRAVITY
2       53      106  # Initial upwards accel.; speed is JUMP_SPEED - GRAVITY
3       53      159  # Initial upwards accel.; speed is JUMP_SPEED - GRAVITY
4       44      203  # Deceleration (upwards speed is decreasing)
5       35      238
6       26      264
7       17      281
8       8       289
9       -1      288
10      -10     278
11      -19     259
12      -28     231
13      -37     194
14      -46     148
15      -48     100  # Downwards speed saturates at -MAX_DOWNWARDS_SPEED
16      -48     52
17      -48     4
18       0      0    # Hit the floor
```

This implementation still allows for repeated jumps; let's fix that too.

⇨ Change the event handling so that jumping twice in close succession does not do anything. More precisely, change the jump handling so that after each jump, the player cannot jump *again* until they have vertically hit a platform (hard blob). Concretely, this means that pressing "up" continuously in the example above would not make a difference until turn **19**. It you're having timing issues, use the `STEP` button in the UI while continuously pressing the "up" key.

Note that this implementation allows you to jump (once!) after falling off a platform (see the `superjump` map). This is a tribute to *Donkey Kong Country*'s "super jumps." Bonus question:

how would you make sure that these super-jumps are not allowed?

Foes [tests 27 -- 35]

The game feels a bit lonely, doesn't it? Let's add a few foes. Here's what we need to change to make this work:

- We need to add support for new types of blobs. We'll do bees, fire, storms, and evil mushrooms.
- We need to do more in collision detection; until now, we only moved the player when it hit a hard blob; now, we also need to act based on collisions (exactly what happens depends on the foe). This requires particular care, since collision resolution might create new collisions between soft blobs (thus, we process soft-blob-against-soft-blob collisions *after* resolving collisions against hard blobs).
- We need to be able to mark blobs as alive or dead; for example, if we bounce on a mushroom, the mushroom is squashed, and it dies. Dead blobs disappear from the board (except for the player, which changes textures). In general (though mushrooms are a special case), if the player touches a foe, they are defeated (and thus the game is over at the end of that turn).

Here are the rules for the four new blobs:

- Fire is the simplest. It does not move on its own (but it's subject to gravity and it stops if it hits a hard blob), and it cannot be killed (unless it falls off the board); if the player touches it, the game is over at the end of that turn. Fire also kills mushrooms, but it doesn't affect bees.
- Storms are simple too; they don't move at all (they are not subject to gravity) and they can't be killed; the only difference is that they blink. They only affect the player; they don't affect other foes. More precisely, their texture starts as `Textures.Storm`, and after `Constants.STORM_LIGHTNING_ROUNDS` rounds it becomes `Textures.Rain`, and then after `Constants.STORM_RAIN_ROUNDS` rounds it goes back to `Textures.Storm`, etc.

Concretely (with 5 rounds of lightning and 10 of rain), it looks like this:

```
Game.__init__
Game.render → Storm
Game.timestep
Game.render → Storm
Game.timestep
Game.render → Storm
Game.timestep
Game.render → Storm
Game.timestep
Game.render → Storm
Game.timestep
Game.render → Rain
Game.timestep
… (Rain appears 10 times total)
Game.render → Storm
Game.timestep
… etc.
```

- Bees are trickier: they travel at a constant vertical speed `Constants.BEE_SPEED` (starting towards the top), until they hit a hard blob (or fall off the board). If they hit a hard blob, their speed is reversed. Bees are not subject to gravity. They cannot be killed (in particular, they are immune too fire). They kill the player, but not other blobs.

- Mushrooms are the most complex: they travel at a constant speed `Constants.MUSHROOM_SPEED` (starting towards the right), and their horizontal speed is reversed every time they hit a hard blob horizontally. Unlike bees, fires, and storms, mushrooms *can* be squished: the mushroom is squished if the player collides with it vertically, from above (that's equivalent to the minimal displacement vector having `y > 0`). Touching a mushroom from the side or from below kills the player.

⇨ **Implement support for fire, storms, mushrooms, and bees.**

Note: Dead blobs (trees and mushrooms touched by fire, squished mushrooms) do not participate in collisions, and are not displayed onscreen (thus `render` should not return them). The player blob is a bit different: if it dies, it stays onscreen, but with texture `Textures.PlayerLost`.

Special items [tests 36 -- 38]

The only significant part that we're missing is collectible items. These items disappear after the player comes in contact with them.

Helicopter

The helicopter blob (`h`) grants the player flying powers. After passing over a helicopter, the player's texture should change to `Textures.Helicopter` (until the end of the game, at which point the usual `Textures.PlayerWon` or `Textures.PlayerLost` should be displayed), and checks making sure that multiple jumps can't happen in mid-air should be disabled. Players don't get bored when they're a helicopter!

➩ **Add support for the helicopter blob.**This enough to play `flappy` and `flappy-bees` !!

Water and boat

The water blob is just like the floor blob, with one exception: when the player is touching water, its sprite should change to `Textures.Boat` but when the player jumps from water or moves horizontally off the water, its texture should revert to the regular player texture. Players don't get bored when they're a boat! Other moving blobs interact with it just like a floor (mushrooms walk on water, and bees change direction when they hit it).

➩ **Add support for the water blob.**

Fireballs

Our last item is the fireball. Upon collecting a Sun blob (`s` , with texture `Textures.Sun`), the player can shoot a total of `Constants.SUN_POWER` fireballs. Fireballs are shot using either the `x` key (to shoot right) or the `z` key (to shoot left). They move at `Constants.FIREBALL_SPEED` towards the right or the left. Pressing both `x` and `z` shoots two fireballs (if the player has only one fireball left, ignore `z` and only shoot towards the right). The starting position of each fireball is exactly the position of the player on the time step when the fireball is shot, *before* the player moves.

Fireballs die against all hard blobs as well as mushroom and trees. If a fireball touches a mushroom or a tree, that blob dies too (bees are immune to fire and fireballs). Fireballs pass through other blobs without affecting them, and are not affected by gravity. There is no limit on how many unused fireballs a player can collect at a time.

➩ **Add support for fireballs.**

Efficient collision detection [test 39]

The algorithm that we implemented in week 1 for collision detection works OK, but it's much too slow once large numbers of blobs are involved. The are two commonly used tricks to make things better:

- Don't simulate blob motion outside of the field of view. This works, though it may make some things feel unnatural. It's good for timing enemy actions to the player's arrival, though. We won't implement this (but feel free to experiment on your own!).

- Improve the collision detection algorithm. Our original implementation considers all pairs of blobs, which is much too slow.

⇨ **Implement a fast collision detection algorithm.**

You are free to use a well-known data structure like a quadtree, or to implement your own solution tailored to this problem (our solution is just over 15 lines of pretty simple code; don't overthink it!). Unlike week 1's implementation, your solution doesn't need to work with arbitrary rectangles — we will only test it with actual game scenes. Concretely, this means that you need to be able to select the `w2-tests-35-many-collisions` map in the UI, and things should be fluid.

Now go explore the `w2-game-*` maps, and enjoy the smoothness of your efficient collision detector :)

Extending the game

We've worked hard to make the game extensible. You can add new game maps to `resources/maps/` (each game map is a simple text file). You can also add new actions (your game receives all keystrokes, no just, `x` , `z` , `left` , `right` , and `up`). Finally, you can add new blob types; you can give them any behavior you want. Some ideas:

- A tree that continuously produces new mushrooms.
- A beanie that lets you jump higher.
- A friend-zapper that turns evil mushrooms into friendly mushrooms; upon touching a friendly mushroom, evil mushrooms turn into friendly mushrooms.
- Smarter fireballs (that you could throw in various directions).
- Moving platforms.
- Platforms that disappear and reappear.
- Smarter enemies.
- Larger or smaller blobs: instead of a `"pos"` , you can include a `"rect"` (`[x, y, w, h]`) in the serialized output.
- A helicopter implementation that points left or right depending on the direction the helicopter is flying (you can use a negative width in the serialized "rect" to ask the UI to reverse a picture).
- An evil twin that follows the player with a 20-cycle delay; the evil twin performs exactly the same actions as the player, but with a delay; touching it causes the player to loose. This forces the player to keep moving and not backtrack.

And if you're feeling motivated, you could even make it a multiplayer game (it should be pretty easy: just handle `A`, `W`, and `D` to move a different "player" blob around)!

⇨ **Implement your own extension of the game.** This could be a different behavior, a new map, etc. There are no tests for this, but we want to see what you came up with at debriefing time (we won't grade you on this; we'll just check that you did implement an extension — it doesn't need to be fancy; something simple will do).

To add a new blob, you'll want to extend the `Textures` class and the `Constants.TEXTURE_MAP` dictionary. You can refer to `resources/emoji_table.txt` for a list of Emoji codes.

If you design a new map, feel free to share it with your classmates and the staff on Piazza! And congrats on completing the lab!

Appendix: blob list

The following table lists all blob types that we ask you will implement in this lab:

Blob	Hard?	Feels gravity?	Character in maps	Textures
Bee			`e`	`e`
Building	✓		`B`	`B`
Castle	✓		`C`	`C`
Cloud	✓		`c`	`c`
Fire		✓	`f`	`f`
Fireball				`F`
Floor	✓		`=`	`=`
Helicopter		✓	`h`	`h`
Mushroom		✓	`m`	`m`
Player		✓	`p`	`p` (normal), `b` (boat), `h` (flying), `bored`, `defeat`, `victory`
Storm	✓		`s`	`s` (storm with thunder), `r` (rain)
Sun		✓	`o`	`o`
Tree	✓		`t`	`t`
Water	✓		`w`	`w`