Ninedraft

Assignment 3
CSSE1001/7030
Semester 1, 2019

Version 1.1.0
20 marks

Due Friday 31st May, 2019, 20:30

1. Introduction

This assignment provides you the opportunity to apply concepts taught throughout the course to extend the functionality of a basic 2d sandbox game, in the style of Minecraft & Terraria.

The main concepts involved are Graphical User Interfaces (GUIs) and object-oriented programming. The assignment tasks are to add features to the game, as described in the requirements below.

You are encouraged to review some similar games, to better understand how this type of game is played, and for inspiration on advanced features. It is better to do this after reading through this document in its entirety.

Because this assignment deals with multiple files, while not required, you may wish to investigate a more sophisticated IDE. A popular option is PyCharm, which is free for students. VS Code, which is also free, is another common option. Please note that these tools have significantly more complex user-interfaces than IDLE, so you may find them a little overwhelming if you are only familiar with IDLE.

2. Overview

2.1. Getting Started

The archive `a3_files.zip` contains all the necessary files to start this assignment. A significant amount of support code has been supplied so that you begin with a simple application that is almost working.

The main assignment file is `app.py`, which contains an incomplete implementation of `Ninedraft`, the top-level GUI application class. The other files are support code which **must not** be edited. `crafting.py` is an exception to this rule, as it must be edited for some tasks.

Initially, you do not need to understand much of the provided code, but as you progress through the tasks, you will need to understand more of this code. You should add code to `app.py` and modify `Ninedraft` to implement the necessary functionality.

You are permitted to create additional files to simplify the separation of tasks (i.e. task1.py, task2.py, etc.), although this is not required. If you do this, `app.py` **must** be the entry point to your application (i.e. running it will run your assignment).

2.2. Pymunk Library

Physics is implemented in the game using the Pymunk library. You will need to install this library in order to implement your tasks for this assignment. Pymunk can be installed by running the included `setup.py`.

3. Assignment Tasks

3.1. Task Overview

This assignment is broken down into three main tasks:

1. The first task involves adding lines of code to clearly marked sections within the main assignment file.
2. The second task involves extending the design to add more interesting functionality to the game.
3. And the third task involves adding sophisticated functionality to further improve the gameplay experience.

For CSSE7030 students only, there is an extra task that involves doing independent research.

In general, as the tasks progress, they are less clearly prescribed and increase in difficulty.

3.2. Task Breakdown

CSSE1001 students will be marked out of 20 and CSSE7030 students will be marked out of 26 based on the following breakdown. Tasks may be attempted in any order, but it is recommended to follow this breakdown, top-down, completing as much as possible of each task before moving on to the next.

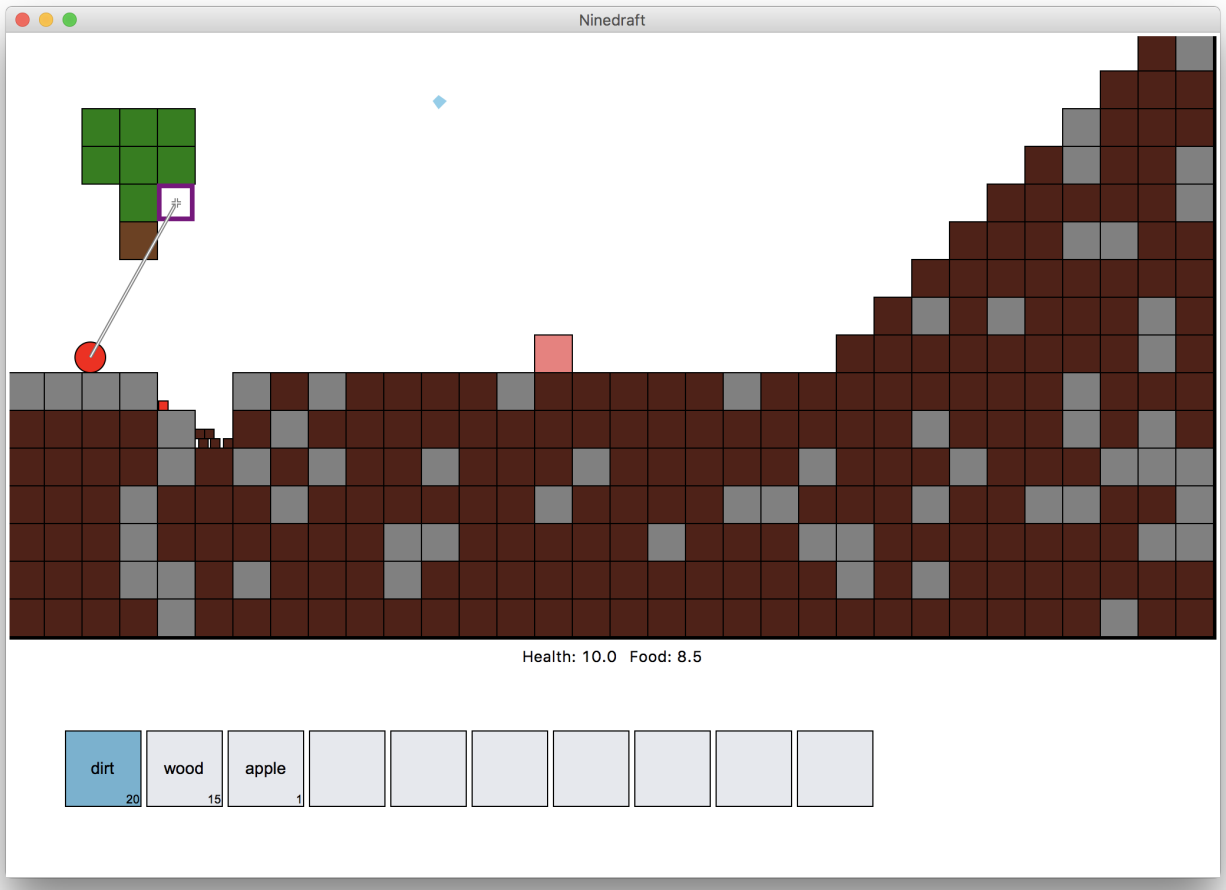
	Sub–Task	Marks
Task 1 *Basic Features*		**9 marks**
	App Class	1 mark
	Mouse Controls	2 marks
	StatusView Class	2 marks
	Basic Items	1.5 marks
	Keyboard Controls	1 mark
	File Menu & Dialogs	1.5 marks
Task 2 *Intermediate Features*		**7 marks**
	More Items	2 mark
	Crafting	3.5 marks
	CraftingTableBlock	1.5 marks
Task 3 *Advanced Features*		**4 marks**
	Mobs	2 marks
	Furnace	2 marks
Post-Graduate Task *Independent Research*		**6 marks**
	Arrow Movement	4 marks
	Interaction with Blocks	2 marks

3.3. Mark Breakdown

For each task, marks will scaled according to the following breakdown.

	Description	Marks
Code Quality	Code is readable. Appropriate and meaningful identifier names have been used. Simple and clear code structure. Repeated code has been avoided.	**15%**
	Code has been simplified where appropriate and is not overly convoluted.	**10%**
	Documented clearly and concisely, without excessive or extraneous comments.	**15%**
Functionality	Components are functional, without major bugs or unhandled exceptions. *Assessed through user testing/playing, not automated testing.*	**60%**

4. Task 1 – Basic GUI



Basic GUI Example

There are a significant number of comments in `app.py` intended to help you complete this task.

4.1. App Class

Write a `main` function that launches the `Ninedraft` GUI. Call this `main` function inside an `if __name__ == ...` block.

Modify `Ninedraft` so that the title of the window is set to something appropriate (i.e. Ninedraft, etc.).

4.2. Mouse Controls

4.2.1. Moving: Target

When the player's mouse is moving over the game world, show the target cursor over the block position they have moused over. If the mouse leaves the game world (either to another widget or out of the window), or moves out of range, hide the preview.

Due to the program structure, once the appropriate tkinter event is bound, the only code that needs be added to achieve this is in `Ninedraft.redraw` (Mouse coordinates for the target cursor are saved when the mouse moves so that `redraw` can use them when it is drawing).

See the `_mouse_move, redraw` methods on `Ninedraft` and the `show_target, hide_target` methods on `GameView` in `game.py` .

4.2.2. Left Click: Attacking (Mining)

Allow the player to attack (mine) their target block by clicking the left mouse button. Code already exists to call the attack method of the player's effective item and to mine the block.

Most of this logic is already implemented. You need to bind the mouse button & retrieve what the block drops when it is fully mined.

Further, when a block is successfully mined, reduce the player's health/food by an amount of your choosing according to the following rules:

- If the player has food (> 0), decrease their food
- Otherwise, decrease their health

See the `_left_click` method on `Ninedraft` .

4.2.3. Right Click: Using or Placing

When the player right clicks, one of two things should happen:

1. If the player is targeting a block, use that block
2. Otherwise, place the active item (see `Item.place`)

Code has been provided that implements this functionality. You only need to bind the appropriate mouse event.

See the `_right_click` method on `Ninedraft` .

See the `__init__, _left_click, _right_click, redraw` methods on `Ninedraft` and the `show_target, hide_target` methods on `GameView` in `game.py` .

4.3. StatusView Class

Define a class named `StatusView` which inherits from `tk.Frame` . This class is used to display information to the player about their status in the game. The `StatusView` 's widgets must:

1. be updated whenever necessary (i.e. when gaining or losing health or food)
2. be laid out *approximately* according to Basic GUI Example
3. contain the following widgets:
    - **Health (first row; left)** A label to display the amount of health the player has remaining, with an image of a heart to the left. The health must be rounded to the nearest 0.5 (i.e. half or whole).
    - **Food (first row; right)** A label to display the amount of food the player has remaining, with an image of a drumstick to the left. The food must be rounded to the nearest 0.5 (i.e. half or whole).

Note: For convenience, you should have a setter method for each of the relevant widgets. i.e. `set_health(health)` , etc.

The `StatusView` class should be added to the application in a frame below the `GameView`.

4.4. Basic Items

One basic type of item is that which drops a block form of itself when placed. For example:

1. When the stone item is placed, it is logical that a stone block should appear
2. Further, when a stone block is mined, it also is logical that it should drop a stone item(s)

The class `BlockItem`, found in `item.py`, represents an item as per #1 above.

For this task, modify the `create_item` function so that it can generate wood & stone items. These should be instances of the `BlockItem` class. E.g.

```
>>> wood = create_item('wood')
>>> wood
BlockItem('wood')
>>> wood.place()
[('block', ('wood,))]
>>> block_id = wood.place()[0][1]
>>> create_block(*block_id)
ResourceBlock('wood')
```

You do not need to modify `create_block`, as the relevant code to create wood & stone blocks already exists.

4.5. Keyboard Controls

4.5.1. Movement

When the player presses the space bar, they should jump into the air. This can be achieved by modifying their velocity.

Set the velocity to something reasonable, that meets the following requirements:

- The y-component must be negative (because computer graphics are drawn with the positive y-axis facing down)
- The x-component must not be zero, but should be different to what it was (i.e. if the player was moving left, when they jump, they should also keep moving left, but at a different speed)
- Double/triple/etc. jumping is allowed, so you do not need to check that the player is on the ground before jumping

4.5.2. Hotbar

When the player presses a number key (1-9, 0), the corresponding item in the hotbar should be selected. If the corresponding item is already selected, it should instead be deselected. Note that 1 corresponds to the first (leftmost) item in the hotbar, and 0 to the last (rightmost), etc.

4.6. File Menu & Dialogs

Implement a menu bar, with a `File` menu. The File menu should have the following entries:

- `New Game`: Restarts the game
- `Exit`: Exits the application

When the player attempts to exit the application, either by the file menu or otherwise, they should first be prompted with a dialog to confirm that they indeed want to quit the application. Further, if the player dies, they should be informed of this with a dialog, and asked if they want to restart the game.

Note: On Mac OS X (and similar), the file menu should appear in the global menu bar (top of the screen).

5. Task 2 – Intermediate Features

For any subclasses you implement, in addition to the required methods listed, you will also need to override any relevant methods from the super class that would raise `NotImplementedError`. You will also need to modify the relevant `create_*` functions to allow your new things/items to be created.

5.1. More Items

Implement the following subclasses of the `Item` class and add them to the game:

5.1.1. FoodItem

Implement a class called `FoodItem` which inherits from `Item`. When a `FoodItem(item_id: str, strength: float)` object is instantiated, it needs to be given an item identifier and a strength. The class must have the following method:

- *get_strength() -> float*: Returns the amount of food/health to be recovered by the player by when used.
- *place() -> float*: Returns an effect that represents an increase in the player's food/health

Add a stack of 4 of these to the starting hotbar, and modify the `create_item` function so that leaf blocks can drop apples (a food item with 2-strength)

When food is placed, increase the player's health/food by an amount equal to the food's strength, according to the following rules:

- If the player has lost food (< maximum), increase their food
- Otherwise, increase their health

Nothing should physically be "placed" in the world. For example:

```
>>> apple = FoodItem('apple', 2)
>>> apple.place()
[('effect', ('food', 2))]
>>> apple.get_strength()
2
```

See `Ninedraft.run_effect`

5.1.2. ToolItem

Implement a class called `ToolItem` which inherits from `Item`. When equipped, tools damage certain blocks faster than the hands, and are able to mine materials that the hands cannot (such as stone from stone blocks). A tool is depleted when its durability reaches zero.

When a `ToolItem(item_id: str, tool_type: str, durability: float)` object is instantiated, it needs to be given an item identifier, the type of tool it will be and the tool's durability. The class must have the following methods:

- *get_type() -> str*: Returns the tool's type.
- *get_durability() -> float*: Returns the tool's remaining durability.
- *can_attack() -> bool*: Returns True iff the tool is not depleted.
- *attack(successful: bool)*: Attacks with the tool; if the attack was not successful, the tool's durability should be reduced by one.

Note: For tools made of a material, such as the stone axe, the `item_id` should be `{material}_{tool_type}`, according to the break tables in `item.py`

For example

```
>>> diamond_axe = ToolItem('diamond_axe', 'axe', 1337)
>>> diamond_axe.get_type()
'axe'
>>> diamond_axe.get_max_stack_size()
1
>>> for _ in range(1295):
...     diamond_axe.attack(False)
```

```
>>> diamond_axe.get_durability()
42
```

5.2. Crafting

The process of turning items into new items is called Crafting. The player can craft items and reorder their inventory/hotbar in a crafting screen. When the player presses the 'e' key, the basic crafting screen should toggle (switch between showing in a new window & hiding). You may assume the player will not interact with the game world while a crafting window is open.

Add the `CRAFTING_RECIPES_2x2` constant to `app.py` with at least three additional recipes of your own, and modify `Ninedraft._trigger_crafting` to open the simple crafting window. Initially, this will only show the inventory & hotbar.

Next, in the file `crafting.py`, complete the `GridCrafterView` class according to the comments. You do not need to modify any other code in this file (`GridCrafterView` is already instantiated in `CraftingWindow._load_crafter_view`).

See `CraftingWindow` & `GridCrafter` in `crafting.py`

5.2.1. Snippets

These can also be found in `a3_files/snippets.txt`

5.3. CraftingTableBlock

Implement a class called `CraftingTableBlock` which inherits from `ResourceBlock`. When used, this block should trigger the crafting table screen. From the block's perspective, this can be achieved by returning an effect from the `use` method. When mined with the correct tool, this block should drop an item form of itself. For example:

```
>>> table = CraftingTableBlock()
>>> table.use()
('crafting', 'crafting_table')
>>> table.get_drops(0, True)
[('item', ('crafting_table',))]
```

Add the 2x2 recipe to create the crafting table to `CRAFTING_RECIPES_2x2`. Further, add the `CRAFTING_RECIPES_3x3` constant to `app.py` with at least three additional recipes of your own.

Lastly, extend the `Ninedraft._trigger_crafting` method to open the crafting table screen.

5.3.1. Snippets

These can also be found in `a3_files/snippets.txt`

6. Task 3 – Advanced Features

6.1. Mobs

Mobs are non-player-characters in the game. They are living (have health) and can move around. Some mobs can harm the player.

To complete this sub-task, in addition to subclassing `Mob`, you will need to add additional blocks & items to the game, including the relevant `create_*` function, and drawing method(s) to the `WorldViewRouter` class. Further, you will need to handle the case where the player attacks (left-clicks) a mob.

6.1.1. Sheep

Implement a class called `Sheep` which inherits from `Mob`.

Sheep move around randomly and do not damage the player.

When attacked, a sheep should drop wool (`wool = BlockItem('wool')`), without taking damage. This wool should be able to be dropped as a block and should be included in at least one crafting recipe.

6.1.2. Bees

Implement a class called `Bee` which inherits from `Mob` .

Bees swarm the player, and individually cause a small amount of damage. They move slightly quicker than the player, and swarm toward the player in a semi-random fashion. If there is a honey block (`honey = ResourceBlock('honey', ...)`) nearby (within 10 blocks), they will instead swarm to the honey.

The player can attack bees, and with simple tools they should individually be destroyed in one hit.

When the player mines a hive block (`hive = HiveBlock("hive")`), 5 bees should spawn and begin to swarm toward the player.

6.2. Furnace

The furnace allows the player access to another form of crafting, called smelting (also known as cooking, baking, melting, drying, or burning). This process involves two items, a fuel source and the item to be heated. For example, the player can cook an apple on wood to produce a cooked apple.

Allow the player to craft a furnace on the crafting table by using a ring of stone (a 3x3 grid, with nothing in the centre). This should yield an item that can be placed as a furnace block, which when used opens the smelting screen, and when mined with a pickaxe drops an item form of itself.

Ensure that the smelting screen shows some sort of icon to represent the smelting — you may use a simple canvas shape in place of the flame shown above.

Lastly, add at least five smelting recipes to create items. You are encouraged to add new items/blocks to make this interesting (i.e. diamond, gold, grain, etc.), although this is not required.

7. CSSE7030 Task – Independent Research

7.1. Advanced Feature

This task involves adding a bow & arrow tool to the game. Add it to the player's starting inventory.

When attacking, the bow should fire a flaming arrow at a trajectory matching the mouse's position relative to the player. For example, if the player is centred at `(200, 200)` , and the cursor is at `(100, 100)` , the bow should fire at a 45 degree angle to the left. The further the mouse is from the player, the stronger the arrow should fire, up to some maximum (i.e. 4 blocks away).

It is intended that gravity should cause the arrow to drop, and not necessarily pass through the position on the cursor. Further, the bow should always fire, regardless of whether the target is in range.

Physics in Ninedraft is implemented in the `World` class using the pymunk library, so to successfully complete this task, you will need to research this library and subclass `World` .

Further, to achieve full marks for this task, the flaming arrow must immediately destroy certain blocks on contact:

- Wood is instantly destroyed and drops nothing
- Hive blocks are instantly destroyed, and drop honey (`BlockItem('honey')`) without spawning any bees.

8. Assignment Submission

Note: There will not be a practical interview for the third assignment.

Your assignment must be submitted via the assignment three submission link on Blackboard. You must submit a zip file, `a3.zip`, containing `app.py` and all the files required to run your application (including images). Your zip file must include all the support code.

Late submission of the assignment will **not** be accepted. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment *well* before the submission deadline. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment. An incorrect version that does not work **will** be marked as your final submission.

In the event of exceptional circumstances, you may submit a request for an extension. See the course profile for details of how to apply for an extension. Requests for extensions must be made **no later** than 48 hours prior to the submission deadline. The expectation is that with less than 48 hours before an assignment is due it should be substantially completed and submittable. Applications for extension, and any supporting documentation (e.g. medical certificate), must be submitted via my.UQ. You must retain the original documentation for a *minimum period* of six months to provide as verification should you be requested to do so.

Change Log

Version 1.1.0 - May 17

It is recommended that you **immediately** update your support code. However, if you began working on the assignment in version 1.0.0, you may submit it with version 1.0.0 of the support code.

Assignment Sheet

- Redistributed some marks within tasks to more accurately reflect difficulty
- Clarified how food/health changes in 4.2. Mouse Controls & 5.1.1. FoodItem
- Clarified terms `target/attack/mine/use/place` in 4.2. Mouse Controls
- Clarified concepts & expanded description in 4.4. Basic Items
- Significantly clarified crafting in 5.2. Crafting, 5.3. CraftingTableBlock, & 6.2. Furnace (added snippets to the first two)
- Fixed typographical errors, annotated relevant types, & added minor clarifications
- Added screenshots of crafting to online view; also available in blackboard
- Reformatted example code to be in the same form as running in IDLE

Support Files

- Standardised parameter names:
  - app.py: `create_block(block_id)`, `create_item(item_id)`
  - block.py: `get_drops(..., correct_item_used)` method of `Block` & its subclasses
- Added comments to 100% of provided source code that requires them
- Moved `WorldViewRouter` to `game.py`, and added parameters to `__init__`; should now be instantiated in `app.py` with `WorldViewRouter(BLOCK_COLOURS, ITEM_COLOURS)`
- Added `Item.get_max_durability` method & added durability display to hotbar
- Fixed items not being picked up when colliding with player
- Mobs:
  - Renamed `creature.py` to `mob.py`
  - Renamed all instances of creature in `world.py` to mob (case-insensitive)
  - Added abstract `Mob` class & example `Bird` class
  - Added `time_delta` & `game_data` parameters to `PhysicalThing.step` to prevent hacky solutions for task3/post-grad
  - Moved `Player` 's health to parent class to share with `Mob`
- Merged `geometry.py` into `core.py` (to try and reduce overwhelming number of files)

- Added optional parameters `collision_types, thing_categories` to `World` in `world.py` to simplify post-graduate task
- Added `crafting.py` with template for crafting view widget