Ninedraft

Assignment 3
CSSE1001/7030
Semester 1, 2019

Version 1.0.0
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 `NinedraftApp`, the top-level GUI application class. The other files are support code which **must not** be edited. Initially, you do not need to understand much of this 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 `NinedraftApp` 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. One way to achieve this is to move `NinedraftApp` to a separate file, such as `base.py` . *Regardless of how you structure your files, the code **must always** be able to be demonstrated by running* `app.py` .

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 & 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**

	Sub–Task	Marks
	App Class	1 mark
	Mouse Controls	2 marks
	StatusView Class	2 marks
	Basic Items	2 marks
	Keyboard Controls	1 mark
	File Menu & Dialogs	1 mark
		7 marks
Task 2 *Intermediate Features*	More Items	2 mark
	Crafting	3 marks
	CraftingTableBlock	2 marks
		4 marks
Task 3 *Advanced Features*	Bees	1.5 marks
	Furnace	2.5 marks
		6 marks
Post-Graduate Task *Independent Research*	Arrow Movement	4 marks
	Interaction with Blocks	2 marks

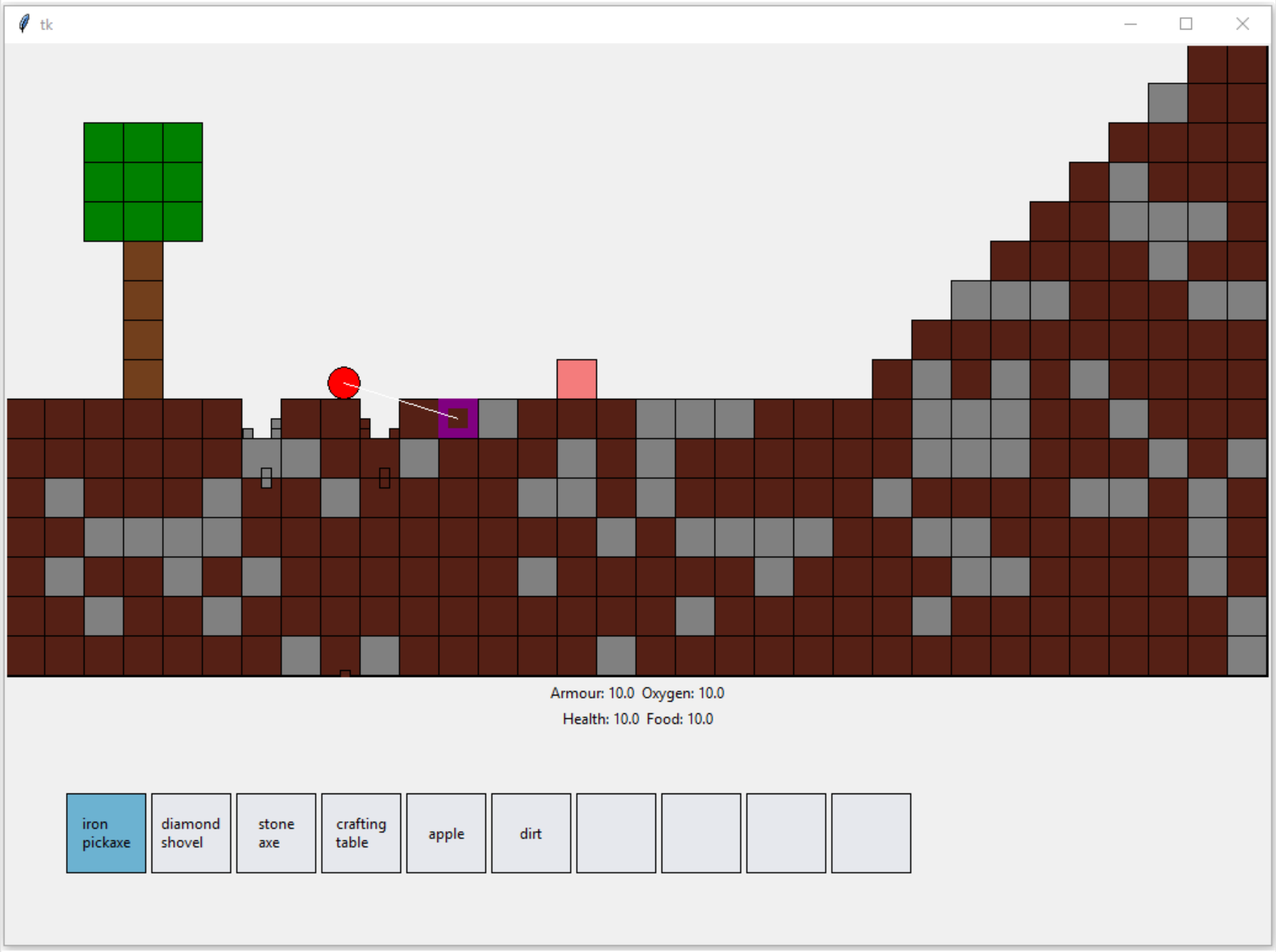
3.3. Mark Breakdown

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

Description	Marks

	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



Armour: 10.0 Oxygen: 10.0
Health: 10.0 Food: 10.0

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 `NinedraftApp` GUI. Call this `main` function inside an `if __name__ == ...` block.

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

4.2. Mouse Controls

Allow the user to attack the target by clicking the left mouse button. While their mouse is being moved the game window (`GameView`), show the target cursor over the block position they have moused over. If the mouse is out of range or leaves the window (`GameView`), hide the target cursor.

Allow the user to use the currently selected item by clicking the right mouse button. When the user is holding a block (dirt, stone, etc.) and right clicks on empty space, which is in range, the block is placed at that location. If the user is instead holding a Item (tool, weapon, bare hand, etc.), then right clicking on the target will attempt to use that item on the target.

See the `_left_click` , `_right_click` and `_mouse_move` methods in `NinedraftApp` .

4.3. StatusView Class

Define a class named `StatusView` which inherits from `tk.Frame` . This class is used to display information to the user 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 – see `Task 1.3 (Status View): Update ...` comments)
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

Modify the `create_item` function so that it can generate wood & stone. These should be instances of the `BlockItem` class, which drops a block form of itself when placed. E.g.

```
>>> wood = create_item('wood')
>>> wood  # Is an object of BlockItem('wood')
>>> wood.place(...)  # Returns an object of ResourceBlock('wood')
```

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 user 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 user 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, strength)` object is instantiated, it needs to be given an item identifier and a strength. The class must have the following method:

- get_strength() -> Returns the amount of food/health the item recovers the player by when used.

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)

5.1.2. ToolItem

Implement a class called `ToolItem` which inherits from `Item`. When equipped, the 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, tool_type, durability)` 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() -> Returns the tool's type.
- get_durability() -> Returns the tool's remaining durability.
- can_attack() -> Returns True iff the tool is not depleted.

- attack(successful) -> 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()  # => 1
>>> for _ in range(995):
...     diamond_axe.attack(False)
>>> diamond_axe.get_durability()  # => 42
```

5.2. Crafting

The process of turning items into new items is called Crafting. The user can craft items and reorder their inventory/hotbar in a crafting screen. When the user presses the 'e' key, the basic crafting screen should toggle (switch between showing in a new window & hiding).

Allow the user to reorder items in their inventory/hotbar by calling & hooking into the relevant methods on the `InventoryView` class. Further, allow the user to move items to and from the simple crafter to craft new items.

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.

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

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

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.

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 = Item('wool')`), without taking damage.

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 grain on wood to produce bread. See details on furnaces & smelting.

Implement the `FurnaceCrafter` class which inherits from `Crafter`. Connect it to your code, such that the player can craft a furnace on their crafting table using a ring of stone (a ring is 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.

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.

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

Any changes to this document will be listed here.