SUPERMARKET ENVIRONMENT SETUP

Authors: Daniel Kasenberg, Teah Markstone, and Matthias Scheutz

Setting: The (autonomous) agent has to perform a shopping task in a supermarket, gathering food items on a shopping list in a cart or basket, paying for them at the cash register and leaving the store with the purchased items. The simulation can be run with keyboard input, or as an OpenAI gym environment connected (via a socket connection) to the Java-based agent you will develop in this course.

Install the following dependencies:

- python 3.7+ (one helpful site is https://docs.python-guide.org/)
- java 1.8 or higher
- pygame 2.0.0+ (can be installed with “pip install pygame”)
- gym (see https://github.com/openai/gym), you can use “pip install gym”
- supermarket simulation (see https://github.com/teahmarkstone/Supermarket-Environment), you can download the zip file from here directly: https://github.com/teahmarkstone/Supermarket-Environment/archive/refs/heads/main.zip
- DIARC.jar (from Canvas: https://canvas.tufts.edu/files/4920740/download?download_frd=1)

Running the simulation with keyboard input

Running the simulation with keyboard input is a good way to get a sense of how the simulation works, in particular, interacting with various objects. Run the following while in the **Supermarket-Environment-main** directory:

```
<python-command> socket_env.py --keyboard_input
```

where <python-command> is the command for running python3.7+ (by default this will just by “python”).

You should now see the graphical window that lets you control the avatar with the keys described below (also available at https://github.com/teahmarkstone/Supermarket-Environment).

Actions and environment dynamics

Below is a table of the actions available in the supermarket environment (left column), and the corresponding key to press in keyboard input to perform the action (right column):

Action	**Input Key**
North	Up arrow
South	Down arrow
West	Left arrow
East	Right arrow
No-op (do nothing)	<none>
Interact with object	Enter/Return
Toggle holding cart	‘c’
Cancel interaction	‘b’
View shopping list	‘l’
View current inventory	‘i’
Get last state observation	(none)

Below is a brief summary description of the actions available and what they do in different contexts:

- North, south, east, west: move the player around on the screen.
- Interact with object: only works when the agent is adjacent to and facing a particular object in the game. **Pauses the player's ability to navigate while a message is displayed on screen; you must perform the "interact with action" or "cancel interaction" effect again to clear this message and continue running the game.** When holding a shopping cart, **the player cannot interact with any objects aside from the cart itself.** Otherwise, effect depends on the object in question:
    - Cart:
        - The user may do this from in front of, behind, or beside a shopping cart.
        - If the player is holding food, this action will put it in the cart.
        - Otherwise, it will display a message but not do anything else.
    - Cart return (bottom left corner of supermarket): if the player is not holding a shopping cart, this will cause the player to pick up a new cart.
    - Shelf:
        - If the player is currently holding food, this command will "put it back on the shelf" (so that the player is no longer holding it).
        - Otherwise, the player will pick up the shelf's food item.
    - Food counters (right side of supermarket)
        - If the player is not currently holding food, this command will cause the agent to pick up the counter's food (the top counter is "prepared foods", the bottom counter is "fresh fish").
        - This is a two-stage interaction, meaning that two consecutive messages are displayed (and thus the "interact with object" action must be performed three times total to complete the transaction and continue the game).
    - Checkout (left side of supermarket)
        - Interacting with the checkout causes the purchase of (1) any unpurchased items that are currently in the player's hand, and (2) any unpurchased items that are in carts that are directly above or below the checkout aisle.
        - Place your shopping cart either directly below or directly above the checkout if you want to purchase its contents.
        - This is a two-stage interaction, meaning that two consecutive messages are displayed (and thus the "interact with object" action must be performed three times total to complete the transaction and continue the game).

- Cancel interaction: clears any messages caused by interacting with an object. For two-stage interactions (checkout and food counters) this will cancel the transaction in question (not obtain the counter's food; not purchase cart contents). For one-stage interactions, this is equivalent to clearing the messages using "interact with object".
- Toggle shopping cart: if the user is currently holding a shopping cart, this lets go of the cart. Otherwise, picks up the cart. Note that to pick up a cart, the user must be behind the shopping cart (adjacent to the handle).
- Display inventory: displays a list of the food (a) that the player is holding, or (b) that is in the player's cart, and the quantities of each food
- Display shopping list: displays a list of the food on the player's shopping list, and the quantities of each food. Only available in keyboard mode; not necessary in autonomous agent mode (since this information is already available in the agent's observation).

General facts

- Objects are solid; you can't walk through walls, shelves, or shopping carts
- When holding a shopping cart, that cart is an extension of you for collision purposes; the cart won't pass through solid objects either
- Shopping lists are randomly generated when the game initializes, and have anywhere between 1 and 12 items
- Shopping carts can each carry a maximum of 12 food items
- Food items must (for now) be picked up and placed in the cart one at a time. If you need three apples, you must do the following three times: go to the shelf, pick up an apple, go to the cart, put the apple in the cart.
- The player can leave their shopping cart anywhere in the store, and can even have multiple shopping carts: any item in any of your carts counts towards your inventory
- You can only *hold* one shopping cart at a time
- Putting food back on the wrong shelf does not mean you'll pick up the wrong food when you interact with that shelf later (if you were in a grocery store in the canned foods aisle looking for soup and saw 15 cans of soup and one apple, would you pick up the apple?)
- In keyboard mode, there's currently no way to visually between unpurchased and purchased items (on the main game screen, or in your inventory/shopping list). We're working on it.

- You can’t leave a cart above or below one checkout, and then expect your food to be purchased when you interact with the other checkout. This should more or less go without saying.
- In the keyboard input mode, you can exit the game with the ESC key, or by leaving through the exit door. You can also Ctrl+C out of the Python program or “X out” of the window.
- In autonomous agent mode, you can exit the game by leaving through the exit door, Ctrl+C in your terminal, or “X out” of the window.

Running the simulation with Java agents

To let your agent control the avatar in the simulation, you first need to start the socket listener:

```
<python-command> socket_env.py
```

Put all of your agent code into a class called **Agent.java** which has the following structure:

```
import com.supermarket.*;

public class Agent extends SupermarketComponentImpl {
      public Agent() {
            super();
            shouldRunExecutionLoop = true;
      }

      @Override
      protected void executionLoop() {
            // this is called every 100ms
            // put your code in here, e.g.
            goSouth();
      }
}
```

Assuming you put the class together with the agent.jar from Canvas in the same directory, then you can compile it in that directory using:

```
javac -cp DIARC.jar Agent.java
```

and you can run using the following (after you first made sure that the socket is running, won't work without it!)

```
java -cp .:DIARC.jar Agent Agent <any other args>
```

(Note that both "Agent" are needed! Use ";" instead of ":" for Windows!)

For debugging you can start "socket_env.py" with a **--file** <path_to_file> flag that will load in the state from the file at the specified path. You can use your keyboard input (with "<yourpythoncommand> socket_env.py **--keyboard_input**") to press the 's' key while running the simulation. If you then enter a filename in the terminal from which they're running the simulation (e.g., "myseed.txt"), then the current state of the game will be saved to "myseed.txt" and can later be reloaded using **--file** flag.

Actions and environment dynamics

Below is a table of the actions available in the supermarket environment (left column), and the corresponding key to press in keyboard input to perform the action (middle column) and action names to be used in the Java code for your agent:

Action	**Input Key (main.py)**	**Java action**
North	Up arrow	goNorth()
South	Down arrow	goSouth()
West	Left arrow	goWest()
East	Right arrow	goEast()

No-op (do nothing)	<none>	nop()
Interact with object	Enter/Return	interactWithObject()
Toggle holding cart	‘c’	toggleShoppingCart()
Cancel interaction	‘b’	cancelInteraction()
View shopping list	‘l’	(None: observation includes complete shopping list)
View current inventory	‘i’	(None: observation includes all necessary information)
Get last state observation	(none)	getLastObservation()

Observations

In autonomous agent mode, you will need to access the state observations which are contained in the Observation class (see below):

- The observation keeps track of the states of the individual objects: **players**, **carts and baskets, shelves**, **registers**, **cart and basket returns**, **counters**, and **checkouts**. There’s an array for each class of object.
    - All objects have a **position** (a 2D array x and y). Objects other than carts, baskets and players also have a width and height. Be careful in how you interpret this; just because a person is colliding with an object doesn’t mean that object.x < player.x < object.x + object.width, etc. Take a look at the collision() and canInteract() methods for various object classes to get a sense of how this actually works.
    - Shelves and counters have a particular food associated with them, represented as a string (“apples”, etc.)
    - Players and carts are facing a particular direction (0=north, 1=south, 2=east, 3=west).
    - Players may be holding a particular food (field “holding_food”, will be null if not holding any food).

    - Players have a shopping list shopping_list of food strings (matching those on particular shelves). There's a corresponding list list_quant of the number of each type of food on the list. For example, if player.shopping_list[i].equals("apples"), then player.list_quant[i] is the number of apples on the shopping list.
    - The Player's curr_cart is the index in observation.carts of the cart the player is holding, or -1 if the player is not holding a cart, same for curr_basket and baskets.
    - Each cart or basket has an owner (who first picked the cart up from the cart or basket return) and a separate variable last_held (who last touched the cart or basket). This probably won't matter in single-agent settings, but would matter in multi-agent games.
    - Each cart or basket has a capacity. This'll just be 12.
    - Each cart or basket has variables contents and contents_quant (same format as a player's shopping list, but for the unpurchased contents of a cart) and purchased_contents and purchased_quant (same format, but for the *purchased* contents of a cart).
- The observation contains a little information about the "interactive stage" of the game, which can presumably help you remember to call interactWithObject() an appropriate number of times:
    - If the user is not currently interacting with an object (i.e., if no interaction message is on the screen), interactive_stage=-1 and total_stages=0.
    - Otherwise, the user is interacting with some object; interactive_stage is the (zero-indexed) current stage, and total_stages will be 1 or 2 depending on if the interaction in question is a zero-stage interaction.
- The observation has some helper methods that may come in handy. You can rely on the individual objects' collision() and canInteract() methods (they're identical to those in the python simulation); be a bit more wary of the other ones. The helper methods are still very much under active development.