Pacman API Specification

Team Houston
Jonathan Wang, Manu Maheshwari, Yanjun Yang,
Tianqi Ma, Youyi Wei, Jiang Lin

We use the command and strategy design patterns for the Pacman game. The character objects (such as Pacman and ghosts) act according to strategies, and use commands to act in the game. All game objects also use the observable-observer design pattern to receive updates about the game state and to handle interactions. The singleton design pattern is used where appropriate for objects that maintain only a single instance.

In this document, we will discuss the use cases for the game. Then we will break down the interfaces and abstract classes in the model. Finally, we will go over the remaining design decisions.

1 Use Cases

There are three groups of use cases for the Pacman game. We will look at the use cases and discuss how they are handled by the API.

- Pacman - how Pacman interacts with the game
  - move in all directions based on user input - Pacman moves by an update strategy, which is called by the update command. The direction can be changed by the switch command.
  - does not go through walls - Pacman checks for collision using the collision command and follows an interact strategy that will handle collisions.
  - teleport through exits - Pacman checks for collision using the collision command and follows an interact strategy that will handle collisions.
  - eat food - Pacman checks for collision using the collision command and follows an interact strategy that will handle collisions.
- Ghosts - how ghosts interact with the game
  - start in the jail - ghosts are initialized in the jail.
  - move based on strategy - ghosts each have an update strategy, which is called by the update command. The ghosts track the open spaces around themselves to determine where to move to.

  - does not go through walls - ghosts check for collision using the collision command and follow an interact strategy that will handle collisions.
  - teleport through exits - ghosts check for collision using the collision command and follow an interact strategy that will handle collisions.
  - eat Pacman - ghosts check for collision using the collision command and follow an interact strategy that will handle collisions.
  - eaten by Pacman when afraid - DispatchAdapter has an afraidTimer field that tracks when ghosts can be eaten by Pacman. The actual ghost-Pacman interaction is identified by the collision command and handled by the interact strategy.

- Game - overall game behavior
  - display game objects - all game objects are observers stored in DispatchAdapter that will be passed to the view.
  - periodically spawn fruit - the fruit has a fruitTimer field that will determine when to add a fruit object to the observers. The fruit tracks the open positions that it can appear at.
  - track game values - DispatchAdapter has score, lives, and dotsLeft fields which will be updated during the game.

2 View

We have a single canvas in index.html where the game and game values will be displayed. In view.js there are functions to set up and run the game and display game objects. The view makes requests to the controller to reset the game as well as update the game state.

- /resetGame - called on page load to setup the game.
- /updateGame - called every 0.1 sec to receive game state. Game objects are drawn onto the canvas based on received data.
- /switchDirection - called based on user input to change Pacman direction from the keyboard and send this information to the Controller.

Based on the returned data, the view draws the game objects onto the canvas. The ghosts and fruit are image files, and the remaining game objects are drawn using canvas functions. Display-only features that do not affect the game operation, such as flashing ghosts, blinking dots, and the Pacman animation are handled exclusively in the view.

3 Controller

The PacWorldController processes GET and POST requests and returns the JSON representation of the DispatchAdapter, which contains all the game objects in the PacWorld. The controller creates the DispatchAdapter and defines the REST endpoints. The controller transfers the information between the game and the view.

These are the supported requests handled by PacWorldController:

- /resetGame - gets the canvas dimension and passes it to the DispatchAdapter; resets game values and resets all game objects to the starting positions.
- /udpateGame - updates the game state with collisions, new locations, removed objects, etc.
- /switchDirection - takes user input from the front end to change the Pacman movement direction.

The controller also gets and sets the port assigned by Heroku for hosting.

4 Model

The model includes the DispatchAdapter, commands, strategies, and game objects.

4.1 DispatchAdapter

The DispatchAdapter communicates with the model and the view. It updates the game object states through commands, utilizing the command design pattern. It contains the observers and values for the game. We put these values here since they are associated with the game. These values include:

- Point dims - dimensions of game map
- static int gridSize - size of grid spaces for game scaling (static for easy access in concrete classes)
- int score - current game score
- int lives - current Pacman lives
- int afraidTimer - time that Pacman can eat ghosts
- int dotsLeft - number of remaining dots
- boolean gameOver - whether the game is over

Methods in the DispatchAdapter include getters and setters for the fields as well as methods to run the game and communicate between objects. All these methods are associated with the game.

- DispatchAdapter() - constructor
- setCanvasDims(Point dims) - set the canvas dimensions.
- getCanvasDims() - get the canvas dimensions.
- getGridSize() - get the game grid size.
- getScore() - get the game score.
- setScore(int score) - set the game score.
- getLives() - get the remaining lives of Pacman.
- setLives(int lives) - set the remaining lives of Pacman.
- getDotsLeft() - get the number of dots left.
- setDotsLeft(int dotsLeft) - set the number of dots left.
- getAfraidTimer() - get the afraid time for the ghosts. When the Pacman eats a big dot, the ghosts will turn to the afraid state. When this timer runs out the ghosts will change back to the regular state.
- setAfraidTimer(int afraidTimer) - set the afraid timer.
- initializeGame() - reset observers and game values and add game objects as observers according to map layout.
- initializeMap() - return 2D array representing layout of game objects.
- updatePacWorld() - update game state, handling object movement and collisions.
- sendCollisionCmd(AGameObject context) - send the collision command with respect to a game object.
- sendSwitchCmd(String switchInfo) - send the switch command with a string specifying what to switch.
- switchDirection(String body) - read user input from body and change Pacman direction.

4.2 Commands

The cmd package has an interface IGameObjectCmd. Three concrete commands, **CollisionCmd**, **SwitchCmd**, and **UpdateCmd** implement this interface. These commands allow game objects to act on the game. The update command calls update strategies. The collision command checks for collisions and calls interact strategies. The switch command switches the states of game objects (Pacman direction and ghost strategy). The hierarchy of this package is shown in Figure 1.

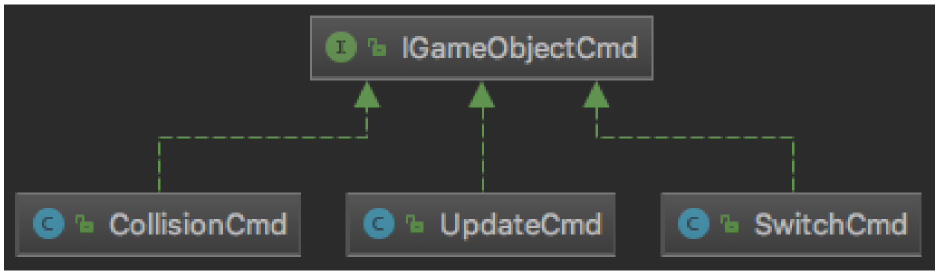

Figure 1: Commands

- IGameObjectCmd - interface for all commands to allow game objects to act on game.
  - execute(AGameObject context) - allows context to act on the game state. The context object is the receiver that executes the command.

4.3 Strategy

The strategy package has two interfaces - IUpdateStrategy and IInteractStrategy.

There are six concrete update strategies that implement the IUpdatesStrategy interface. One of these is the **PacmanUpdateStrategy**. Four are various ghost attack strategies (**GhostChaseStrategy**, **GhostTrapStrategy**, **GhostTrailStrategy**, and **GhostRandomStrategy**) and the last is the **GhostAfraidStrategy**.

Two concrete interact strategies implement the IInteractStrategy interface. The **PacmanInteraction** strategy handles Pacman interactions with food, walls, and exits, and the **GhostInteraction** strategy handles ghost interactions. All Pacman-ghost interactions are handled by GhostInteraction.

The hierarchy of this package is shown in Figure 2.

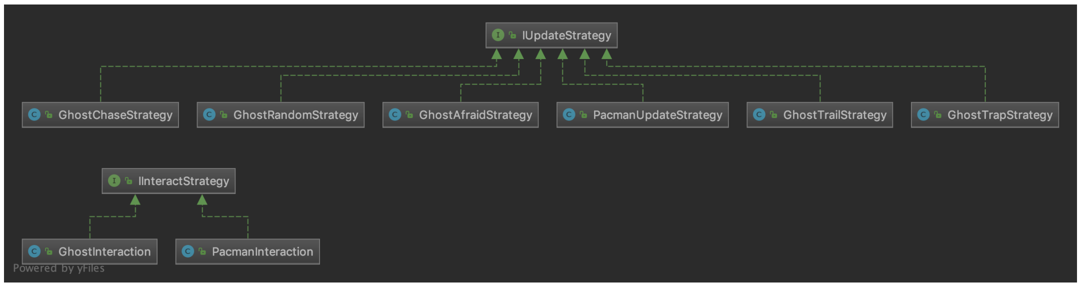

Figure 2: Strategies

- IUpdateStrategy - interface for strategies that update individual game objects.

  - getName() - return update strategy name.
  - update(AGameObject context) - updates the state of the context object.

- IInteractStrategy - interface for strategies that handle interaction between objects.
  - getName() - return interact strategy name.
  - interact(AGameObject src, AGameObject dest) - handles interaction between src and dest objects.

4.4 Game Objects

The gameobject package has an abstract class AGameObject. Two abstract classes, ACharater and AFood, extend AGameObject. ACharacter has two concrete classes: **Pacman** and **Ghost**. AFood has three concrete classes: **BigDot**, **SmallDot**, and **Fruit**. AGameObject has two concrete classes: **Exit** and **Wall**. The abstract classes and the concrete classes are shown in Figure 3.

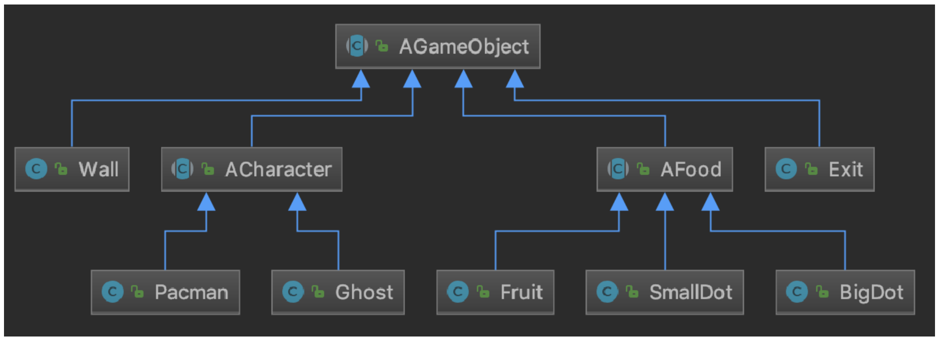

Figure 3: Game Objects

Several of the abstract and concrete classes do not have additional fields or methods from their parent class, but having the added class hierarchy simplifies object creation, and allows for extensible behavior. Each concrete class constructor sets the object size based on the DispatchAdapter gridSize. This allows for easy scalability with just one variable change.

- AGameObject - all game objects have a location, type, and size. Therefore, these fields are put here. All game objects share the following methods:
  - AGameObject(Point location, String type, int size) - constructor.
  - getType() - return the type of the object.
  - getLocation() - return the location of the object.
  - setLocation(Point location) - set the location of the object.
  - getSize() - return the size of the object.

- update(Observable obs, Object o) - update game object by executing o as a command.

- ACharacter - all characters extend AGameObject and have a velocity, updateStrategy, and interactStrategy. They all share these additional methods handling strategies, locations, velocities, collisions, etc.:
  - ACharacter(Point loc, String type, Point vel, IUpdateStrategy updateStrategy, IInteractStrategy interactStrategy, int size) - constructor.
  - getInteractStrategy() - return the interaction strategy of the object.
  - getUpdateStrategy() - return the update strategy of the object.
  - setInteractStrategy(IInteractStrategy interactStrategy) - set the interaction strategy of the object.
  - setUpdateStrategy(IUpdateStrategy updateStrategy) - set the update strategy of the object.
  - getInitialLoc() - return the starting position of the object.
  - getVel() - return the velocity of the object.
  - setVel(Point vel) - set the velocity of the object.
  - collision(AGameObject gameObject) - handle collision against gameObject. We only need to handle collisions in this class since all collisions occur with an ACharacter.
- AFood - all food extends AGameObject. This abstraction layer is used for easier collision checking and possible extensibility, if new behaviors are added for food objects. It has an additional method getPoints() to record the point value of the food.
  - AFood(Point loc, String type, int size, int points) - constructor.
  - getPoints() - return the points.

5 Other Design Decisions

Besides those design decisions mentioned above, there are a few more design decisions that are associated with multiple packages or a concrete class, so we will discuss them here.

- Pacman - when switching directions, we check for wall collision in the new direction. If it will hit a wall, the switch will be queued until the next open space and Pacman will continue moving in the original direction. We add some fields in Pacman and logic in the update and collision commands to handle this behavior.
- Ghosts - track open spaces around themselves to determine where they can move in the update strategies. This is maintained as a list in the ghost and is updated on wall collision.

- Fruit - tracks open spaces on the map to know where the fruit can spawn at. The fruit also has a fruitTimer that determines when the fruit spawns.
- Exit - contains a field determining where objects appear at when going through an exit.