Pac Man API Document

Team DragonBall

For this game, our team used the MVC programming paradigm to structure our code. We also used strategy design pattern, command design pattern and singleton design pattern to support the logic that flows within this game.

1. Controller

The PacmanController transfers the data between the game and the view. It will create and interact with the DispatchAdapter. It also creates the endpoints and can process GET/POST requests.
We have the following requests:

- /restart – This endpoint resets the game to the initial state by calling the init() method in the DispatchAdapter.
- /update – This endpoint gets the user key command and passes the command to the DispatchAdapter to update the moving objects, including man, ghosts, and pacs.
- This function returns the assigned port for Heroku. If heroku-port isn't set, the default port will be returned.

```
static int getHerokuAssignedPort()
```

2. Model

Model contains most of the logic in this game. There are five packages inside: Command, Ghost, Man(the object that the player controls), Pac(the dots in the maze path) and Wall.

2.1 Command

The IPacmanCmd is an interface, and updateGhostCmd is the concrete implementation of IPacmanCmd.

IPacmanCmd

- The interface is used to pass commands to a moving object.
- A parameter is a moving object.

```
public void execute(MovingObject context)
```

UpdateGhostCmd

- This command is the implementation of the interface IPacmanCmd,
  It can be used to update the strategy of a ghost object.
- private field of ghost strategy

```
private IUpdateStrategy ghostStrategy
```

- This function updates the strategy of the ghost. The parameter strategy is the desired strategy to be set.

```
public updateGhostCmd(IUpdateStrategy strategy)
```

2.2 Ghost

The Ghost is a concrete class that extends the MovingObject abstract class to manage all the moving information. It also implements the PropertyChangeListener interface and Typeable interface. To handle all the operations on ghost objects, the class has the following methods and fields:

- The five parameters specify the properties of the ghost upon initialization: loc is the location of the ghost, vel is the initial velocity of the ghost, dir is the Point object that specifies the ghost's direction, radius is an integer and can be used to detect collision, and strategy is an IUpdateStrategy object that specifies the ghost's strategy.

```
public Ghost(Point loc, int vel, Point dir, int radius, IUpdateStrategy stra
tegy)
```

- Return the type of the object.

```
public String getType()
```

- Set the strategy of the object. The parameter updateStrategy is the strategy to be set.

```
public void setUpdateStrategy(IUpdateStrategy updateStrategy)
```

- Returns the update strategy.

```
public IUpdateStrategy getUpdateStrategy()
```

- Checks if the ghost is collided with another moving object. The parameter object is a MovingObject object and will be checked to see if it has collided with the ghost.

```
public boolean isCollidedWith(MovingObject object)
```

- This is the property change event handler. The ghost is listening for a property change. The parameter evt is the property change event.

```
public void propertyChange(PropertyChangeEvent evt)
```

2.3 Man

The Pacman, similar to a ghost object, is a concrete class that extends the MovingObject abstract class to manage all the moving information. It also implements the PropertyChangeListener interface and Typeable interface. To handle all the operations on man objects, the class has the following methods and fields:

- The five parameters specify the properties of the Pacman upon initialization: loc is the location of the Pacman, vel is the initial velocity of the Pacman, dir is the Point object that specifies the Pacman's direction, radius is an integer and can be used to detect collision, and strategy is an IUpdateStrategy object that specifies the Pacman's strategy.

```
public Man(Point loc, int vel, Point dir, int radius, int lives)
```

- Return the type of the object.

```
public String getType()
```

- Set the lives of the Pacman. The parameter lives is an integer, meaning the lives to be set.

```
public void setLives(int lives)
```

- Return the lives of the object.

```
public int getLives()
```

- Reduce the lives of the object.

```
public void reduceLive()
```

- Set the score of the Pacman. The parameter score is an integer, meaning the score to be set.

```
public void setScore(int score)
```

- Return the total score.

```
public int getScore()
```

- Set the level of the object.

```
public void setLevel(int level)
```

- Return the level of the object.

```
public int getLevel()
```

- Add the score of the object.

```
public void addScore(int newScore)
```

- Set the game context of the object.

```
public void setGameContext(GameContext gameContext)
```

- Return the game context of the object.

```
public GameContext getGameContext()
```

- This is the property change event handler. The Pacman is listening for a property change. The parameter evt is the property change event.

```
public void propertyChange(PropertyChangeEvent evt)
```

- Add the eaten number of the object.

```
public void addEatenNum()
```

- Initiate the eaten number of the object.

```
public void initEatenNum()
```

- Return the eaten number of the object.

```
public int getEatenNum()
```

- Add the level of the object.

```
public void addLevel()
```

- Return the next level of the object.

```
public boolean getNextLevel()
```

- Set the next level of the object.

```
public void setNextLevel(boolean flag)
```

- Reset the location and direction of the man

```
public void resetMan()
```

2.4 Moving Object

This class will be extended by ghosts and men, and it manages all the moving information.

- The four parameters specify the location, velocity, direction, and radius of the moving object, which could be the ghost or the man. The five parameters specify the properties of the moving object upon initialization: loc is the location of the object, vel is the initial velocity of the object, dir is the Point object that specifies the object's direction, radius is an integer and can be used to detect collision, and strategy is an IUpdateStrategy object that specifies the object's strategy.

```
public MovingObject(Point loc, int vel, Point dir, int radius)
```

- Use binary search to detect moving object's collision with vertical walls and horizontal walls. Firstly, we use the binary search algorithm to find the closest wall to the moving object by

comparing their x coordinates, then we detect if the wall and the moving object collide or not. verticalWalls and horizontalWalls are walls' coordinates.

```
private boolean isCollisionExist(ArrayList<Wall> verticalWalls, ArrayList<Wa
ll> horizontalWalls)
```

- This function will move the object n steps forward in its direction, and it will stop once it collides with the wall. steps mean the steps that the object plans to go. verticalWalls and horizontalWalls are walls' location used to be passed into isCollision() function for detecting if the object and walls collide.

```
private void goToNextLoc(int steps,ArrayList<Wall> verticalWalls, ArrayList<
Wall> horizontalWalls)
```

- This function will move the object according to the command made by the user. If the object is able to change its direction in this frame, then it will go for d steps based on original direction, and vel - d steps based on new Direction. If the object is not able to change its direction, it will try to go vel steps until it hits a wall.

```
public void move(Point newDir, ArrayList<Wall> verticalWalls, ArrayList<Wall
> horizontalWalls)
```

- This function will try to change the direction of the object. If the object can change its direction based on its own velocity, then return the steps it should go before it changes its direction. Note that this function does not update the object's data. newDir is the new direction for the object. verticalWalls is the vertical walls. horizontalWalls is the horizontal walls. The function returns the steps the object should go before it can change its direction.

```
private int tryChangeDir(Point newDir, ArrayList<Wall> verticalWalls, ArrayL
ist<Wall> horizontalWalls)
```

2.5 Strategy

We have four different ghosts with drastically different strategies in the game, which will all be implemented through several strategy classes described in this section. The strategy package has two interfaces - IUpdateStrategy and ICollisionStrategy. There are seven concrete update strategies that implement the IUpdatesStrategy interface. They are various ghost attack strategies (**AfterDieStrategy**, **BeforeBirthStrategy**, **ChaseManStrategy**, **DarkBlueStrategy**, **FlashingStrategy**, **RandomWalkStrategy** and **StayAwayStrategy**). The hierarchy of this package is shown in Figure.

IUpdateStrategy

The interface will be implemented by all the ghost strategies. The only interface function. It will update the ghost object (context)'s status according to concrete implementation.

```
public void update(Ghost context, PropertyChangeListener[] objects)
```

Ghost Strategies

- Before Die Strategy
  If Pac-Man collides with a dark blue or flashing ghost, the ghosts become two eyes and travel quickly to the square box in the middle of the screen, which is the spawning zone, to become a brand new colorful ghost (which is non-blue).
- Before Birth Strategy
  This strategy will allow three ghosts to move around the initial box region where they were born before we start the game. Ghosts will get out of the box region one by one in a time interval of several seconds after we click the start button.
- Chase Man Strategy
  This strategy is the initial strategy for ghost 3 and ghost 4. They will chase after the man using DFS strategy.
- Random Walk Strategy
  This strategy is the initial strategy for ghost 1 and ghost 2. They will not chase after the ghost, instead, they walk in a random route.
- Stay Away Strategy
  The ghost with this strategy will travel away from the ghost after the Pacman eats an energizer, and at the same time, the color of the ghost will turn into dark blue.
- Flashing Strategy
  This strategy will be called when the ghost is about to recover and return to normal status. The Man will not die when meeting with the flashing ghost.

ICollisionStrategy

The interface will be implemented by all the collision strategies.

2.6 DispatchAdapter

The DispatchAdapter communicates with the model and the view. It will load the map, add all the walls, pacs, ghosts and man. The update to all of the objects also takes place here. We put these values here since they are associated with the game.

- The PropertyChangeSupport object

```
private PropertyChangeSupport pcs
```

- The ArrayList that stores all the vertical walls.

```
private ArrayList<Wall> verticalWalls
```

- The ArrayList that stores all the horizontal walls.

```
private ArrayList<Wall> horizontalWalls
```

- The ArrayList that stores all the pacs.

```
private ArrayList<Pac> pacs
```

- The ArrayList that stores all the ghosts.

```
private ArrayList<Ghost> ghosts
```

- The Pacman object.

```
private Man man
```

- The total number of ghosts.

```
private int numOfGhost
```

- The initial velocity of ghosts.

```
private int initialVel
```

- The initial strategy of ghosts.

```
private IUpdateStrategy initialStrategy
```

- This function first parses the map, then adds the vertical and horizontal walls to the local fields for later processing. The parameter is a JsonObject that contains the map information, and it will be parsed by the function.

```
private void parseMapJson(JsonObject jsonObject)
```

- This function Initializes the map, while adding all the walls, pacs, ghosts and man as listeners.

```
public void init()
```

- This function updates the moving objects, including man, ghosts, and pacs. The parameter dirCmd is an integer that corresponds to the key pressed in the front-end. This function returns the array of PropertyChangeListeners.

```
public PropertyChangeListener[] update(int dirCmd)
```

- This function checks all pacs and remove the eaten pacs.

```
private void clearEatenPac()
```

- This function checks if the game status has changed. If so, the game objects will be updated.

```
private void checkGameStatus()
```

- This function adds a listener to pcs. The parameter is a PropertyChangeListener that will be added to pcs.

```
private void addListener(PropertyChangeListener pcl
```

2.7 Typeable

The Typeable is an Interface. Game objects such as Ghosts, Pacman, Pac, are all concrete implementations of the Typeable interface. This interface can be used to get the types of objects.

- Get the type of the object. The returned value is a string indicating the type of the object.

```
String getType()
```

3. Use Cases

3.1 Pacman

- Pacman moves straight normally.
- Pacman stops when colliding with a wall if it hasn't been given instructions of turning beforehand.

- Pacman changes direction and keeps moving in the next opening based on the last key pressed.
- Pacman will gain points when it eats normal Pac-dots.
- Pacman will gain more points when it eats bonus dots.
- Pacman will gain more points and be empowered thus it can hunt and eat ghosts in the future a few seconds when it eats power dots.
- Pacman will be eaten and die if it collides with ghosts unless it is empowered.

3.2 Ghost

- Ghosts will move around in the spawning zone and start chasing the Pacman when the game begins.
- Two ghosts will chase after Pacman according to the chase man strategy.
- Two ghosts will move randomly according to the random strategy.
- Ghosts will become blue and turn around to move away from Pacman after Pacman eats power dots, and they will turn into flashing state after several seconds before they return to the normal state.
- Ghosts will transform into eyes and travel back home when the ghost is eaten by Pacman.

4. Design Decisions

4.1 Ghost Strategies

To ensure a more challenging gameplay, we designed 4 different strategies for 4 different ghosts. The ghost with Ambush Strategy starts chasing the Pacman when it's close enough to it. The ghost with Intersect strategy tries to find a way to outflank the Pacman. The ghost with random walk strategy walks around the map randomly. The ghost with ChaseManDecision hunts down the Pacman by finding the shortest path between them. When all the pacs are eaten, the game proceeds to the next level, with the ghosts having an increased velocity(+5). Besides, when the Pacman eats an energizer, the time that ghosts stay in dark blue mode and flash will be shortened, meaning that the Pacman has less time to hunt ghosts.

4.2 Fruits

We have 5 distinct kinds of fruits: banana, durian, strawberry, pineapple and watermelon. Fruits will appear at the Pacman's spawning point every 7.5s, and will disappear if the Pacman doesn't eat it within 7.5s.

4.3 Death

When the pacman dies, its life will be set to -1, the user need to click "start" button to restart the game.

5. User Extensibility

5.1 Map Selection

User can choose a map from the drop-down menu on the left. In this version, we have two maps: DragonBall and Basketball.

5.2 Map Editor

By clicking the “map editor” button, user can enter the map editor.
In the map editor, user can draw walls, pac dots, large pac dots (energizer), teleportation tunnel, and ghost walls (the door for ghost). User can also set specific color for walls. The design should follow the rule that the wall near the ghosts’ birthplace is immutable. The ghost can only reach the place where initially there are pac dots.
User can also load a local map(JSON file) or save the designed map in the map editor.