Pac-Man *-- by Team Quicksilver*

Team members

Team Lead: Bill Huang
Tech Lead: Jerry Zhang
Doc Lead: Zhengtong Liu
Frontend Developer: Jeff Zhou, Yang Zhou
Backend Developer: Lou Jianing, Zhen Gao

Interfaces and Abstract Classes

Package: model.board

Interface: IPacmanBoard

```
/**
 * Get the 2D array of the board.
 * Each element in the array represent an element on the board.
 *
 * @return 2D array of the board
 */
int[][] getBoard();

/**
 * Get current score for the pacman.
 *
 * @return current score for the pacman.
 */
int getScore();

/**
 * Tick the game and update board position and status.
 * Run each update.
 */
void tick();

/**
 * Get current level for the game.
 *
 * @return the current game level
 */
int getLevel();
```

Package: model.command

Interface: IUpdateCommand

```
/**
 * Execute certain command on the ghost.
 *
 * @param ghost ghost to execute command
 */
void execute(Ghost ghost);
```

Interface: IUpdateCommandFactory

```
/**
 * Make the IUpdateCommand instance given specified type.
 *
 * @param type type of the command
 * @return IUpdateCommand instance
 */
IUpdateCommand make(String type);
```

Package: model.object

Abstract Class: AConsumableObject

```
/**
 * Get the score of this object when pacman eats it.
 *
 * @return the score of the object
 */
abstract int getScore();
```

Interface: IObject

```
/**
 * Get the location for the object.
 *
 * @return the location of the object
 */
Point getLocation();

/**
 * Get the type of the object.
 *
 * @return type of the object
 */
ObjectType getObjectType();
```

Enum: ObjectType

```
 PACMAN,
GHOST,
WALL,
BISCUIT,
FRUIT,
LARGE_DOT,
QUICKSILVER_CHARM
```

Package: model.strategy

Interface: IStrategyCommand

```
/**
 * Get the name of the strategy.
 *
 * @return name of the strategy
 */
String getName();

/**
 * Update the state of a IObject.
 * The state including location,
 *
 * @param object object to be updated
 */
void update(IObject object);
```

Interface: IStrategyCommandFactory

```
/**
 * Make the IUpdateStrategy Instance given specified type.
 *
 * @param type type of the strategy
 * @return IUpdateStrategy instance
 */
IUpdateStrategy make(String type);
```

Package: util

Class: JsonStatusResponse

```
 private boolean success;
private Object data;
private String message;

/**
 * Constructor of JSONStatusResponse.
 *
 * @param success if the response is success
 * @param data    data of res
 * @param message resp message
 */
public JsonStatusResponse(boolean success, Object data, String message) {
    this.success = success;
    this.data = data;
    this.message = message;
}
```

Design Patterns

Command Design Pattern

We utilize the command design pattern to execute commands on ghosts, such as turning dark blue and then flashing, in order to separate the state changing of ghosts with the implementation of the dispatch adapter. With this pattern, the dispatch adapter can only send commands to ghosts via property change instead of dealing with the shift in ghosts' state.

Strategy Design Pattern

Objects in the Pac-Man game have different movement strategies, so we leverage strategy design patterns to maintain code robustness. Specifically, walls, dots, and fruits have a static strategy because they won't move in the game. However, the Pac-Man and ghosts can move on the game board. Each ghost also has different moving modes, such as chasing, patroling, scattering, etc. These moving strategies are all practices of strategy design patterns.

Factory Design Pattern

To deal with different moving strategies and update commands, we utilize the singleton factory design pattern to unify the creation of these class instances. With this design pattern, we can extend our app functionality more effortless in the future.

Singleton Design Pattern

Some classes should be initialized once and be reused for the next time, such as the factories. We utilize the singleton design pattern to ensure that only one such class instance exists and implement the getInstance() method to share the public singleton class object.

Use Cases

Frontend Endpoints are listed in the next section

- Starts the game
When user starts the game, a request is sent to notify the backend that it can starts to periodically update the specified board. For related Backend use cases, refer to Backend 1.3
- Gets the most up-to-date (position & status wise) game board
The returned game board state will be rendered and displayed in the frontend so user can play the game. For related Backend use cases, refer to Backend 1.1
- Resets the game after GAME OVER before user starts a new game
Before user starts a new game, the frontend will notify the backend to reset user's game board so it is ready for the new game. For related Backend use cases, refer to Backend 1.2
- Makes Pac-Man moves in user specified direction
When user press a direction key, a request will be sent to backend to make Pac-Man on user's game board adopts corresponding update strategy (strategy that makes it move in specified direction). For related Backend use cases, refer to Backend 1.5
- Set the number of ghosts (Min 3, max 8 ghosts)
Notify the backend to set number of ghosts on user's game board to user specified value. For related Backend use case, refer to Backend 1.4.
- Landing. Initialize a game board : For related Backend use cases, refer to Backend 1.6
When user open the site, a request is sent automitically to create a new game board for the user. A boardId will be included in the response and frontend will record boardId, so in each subsequent requests it will be consistently referring accessing the same board. Distinguishing game boards with boardId allows multiple isers accessing the app and plays spontaneous.

Backend (IF : THEN)

1. Controller and Adapter

1.1. Frontend requests the game board : return current board (position & status) to frontend
Adapter method first retrieves the PacmanBoard object with specified boardId and call getScore(), getLevel(), and getBoard(), on the baord to get its attributes score (indicate scores user had achieved in current game), level (indicate the level user is playing on) and board (represents the whole board and positions of each objects on the board).

```
@param String boardId
@return json response
```

1.2. Frontend requests a reset : return the default game board to frontend
Adapter method first retrieves the PacmanBoard object with specified boardId and make call related functions on the baord to restore the board to the default state (respawn Pac-Man, ghosts and dots at their initial coordinates).

```
@param String boardId
@return json response
```

1.3. Frontend starts the game : backend invokes the game board to updates
Adapter method first retrieves the PacmanBoard object with specified boardId and make call tick() method on the baord to start updating the board state.

```
@param String boardId
@return json response
```

1.4. Frontend requests to change number of ghosts : backend set number of ghosts of specified board
Adapter method first retrieves the PacmanBoard object with specified boardId and make call related functions on the baord. Refer to Backend 3.3 for details about the implementation in the model.

```
@param String boardId
@return json response
```

1.5. Frontend requests to make Pac-Man move in specified direction : backend make changes to Pac-Man accordingly
Adapter method first retrieves the PacmanBoard object with specified boardId and make call related functions on the baord. Refer to Backend 2.7 for details about the implementation in the model.

```
@param String boardId
@param String direction
@return json response
```

1.6. Frontend request to initialize a game board : backend initialize a game board
A new PacmanBoard object will be initialized and stored in the adapter. The new board has a unique synthetic key, boardId (String), and its boardId will be included in the response. All objects on the board will be invoked. Refer to Backend 2.1, 3.1, 4.1, 4.2 for details about the implementation in the model.

```
@return JsonStatusResponse response
```

The followings are all concrete classes. A method may not be specified for each use case as these classes are still under development

2. Pac-Man:

2.1. New board initialized : Pac-Man spawns on the game board
When a new board is initialized (first-time or higher level), Pac-Man will spawn at a fixed location on game board. Pac-Man's remaining lives will be set to 3. This operation is done in PacmanBoard constructor
2.2. Pac-Man has 0 lives : GAME OVER, level failed
If the game is ongoing and Pac-Man's remaining lives become negative, game is over and updates on the board will stop.
2.3. For each board state update : Pac-Man moves according to its update strategy
Pac-Man is a derived class of PropertyChangeListener, so it is updated by calling firePropertyChange() method. Pac-Man's coordinates is updated, and it moves one unit in the direction specified by its updateStrategy, if not blocked by a wall, each update. The last coordinates Pac-Man is at before the update will be set to empty
2.4. Pac-Man go into the exit on right side of the map : teleport to the left side exit
If during a state update, Pac-Man's coordinate on the board is marked as right exit and Pac-Man is moving to right, Pac-Man will be teleported to the coordinate that represents the left exit.
2.5. Pac-Man go into the exit on left side of the map : teleport to the right side exit
If during a state update, Pac-Man's coordinate on the board is marked as left exit and Pac-Man is moving to left, Pac-Man will be teleported to the coordinate that represents the right exit.
2.6. Pac-Man eats all the dots and has lives remain : GAME OVER, level completed
If the game is ongoing and there is no more dots on the board, game is over and updates (by tick() method) on the board will stop. Increment level attribute of current PacmanBoard by 1, so that a new game on this board will be more difficult.
2.7. Pac-Man receives instruction to change direction : update strategy is changed
Pac-Man's update strategy will be changed according to the input string.

3. Ghost:

3.1. New board initialized : A specified number (min 3, max 8) of ghosts spawn in the middle of the game board
When a new board is initialized (first-time or higher level), all ghosts will spawn at a designated area in the middle of the game board. This operation is done in PacmanBoard constructor
3.2. For each board state update : Each Ghost moves in a unique behavior
Ghost is a derived class of PropertyChangeListener, so it is updated by calling firePropertyChange() method. Ghost's coordinates is updated, and it moves according to its updateStrategy. Ghost has one of the four movement patterns: towards Pac-Man, randomly, away from Pac-Man or roam along a fixed route. The ghosts' movement speed is set when they are initialized, and speed has a positive correlation with game level
3.3. Set the number of ghosts : number of ghost is set to specified number
Before the game starts, number of ghosts may be modified. The maximum number of ghosts allowed is 8. When user changes the number *N* of ghosts, the ghost store of PacmanBoard object will be reinitialized to size *N*, and *N* new ghost objects will be added to the store. Each ghost will be assigned a random initial behavior.

4. Consumables

4.1. New board initialized : spawn 240 small dots on the game board
When a new board is initialized (first-time or higher level), all 240 small dots will spawn at fixed locations throughout the game board, i.e. 240 coordinates on board in PacmanBoard object will be set to the integer that represents the small dot. This operation is done in PacmanBoard constructor
4.2. New board initialized : spawn 4 large dots on four corners of game board
When a new board is initialized (first-time or higher level), all 4 large dots will spawn at fixed locations on four corners of the game board, i.e. 4 coordinates on board in PacmanBoard object will be set to the integer that represents the large dot. This operation is done in PacmanBoard constructor
4.3. After every *N* board state updates : fruit spawn on random empty location on the game board
After every *T* board state updates, a fruit will spawn at a random empty location on the game board. One coordinate on board in PacmanBoard object will be set to the integer that represents fruit. Value *T* is determined by level.
4.4. At Most once during the game, randomly : Quicksilver charm spawn on random empty location on the game board
Every board state update a random number generator will produce a number, and if number equals a pre-set value *V*, a quicksilver charm will spawn at a random empty location on the game board. One coordinate on board in PacmanBoard object will be set to the integer that represents Quicksilver charm. In each game, the charm will only spawn once. To prevent the charm from spawning again, *V* will be set to a value outside of the range of random number generated after charm spawns for the first time.

5. Pac-Man ghost interaction:

5.1. Pac-Man collides with a NON-dark blue/NON-flashing ghost : Pac-Man loses 1 life
Pac-Man's remaining life field decrement by 1. Freeze the game (i.e. pause board state update) for a few seconds. Then if Pac-Man's remaining life is less than 0, GAME OVER and refer to Backend 2.2. Else, respawn Pac-Man at a random empty coordinate on board.
5.2. Pac-Man collides with a dark blue/flashing ghost : the ghosts become 2 eyes and quickly moves to the square box in the middle of the game board
The integers that represents this particular ghost on the board will be replaced by one that represents a pair of eyes. Change the update strategy of this to "move quickly back to ghost respawn area". After the ghost arrives at respawn area, change the integer back to one that represent a normal ghost and reassign a update strategy to this ghost.
5.3. Pac-Man collides with a dark blue/flashing ghost : score increment by 200 (1st time in current period), 400 (2nd), 800 (3rd), 1600 (4th)
Increment score attribute of the current PacmanBoard by a value "S". Double the value of "S". Refer to Backend 5.2, 6.3, and 7.1 for more use cases related to this one.

6. Pac-Man consumables interaction:

6.1. Pac-Man eats small dots : small dots disappear and score increment by 10
When Pac-Man moves onto the coordinate of a small dot, the dot will be considered eaten. No modification on board is necessary here as it is done by update of Pac-

Man coordinate. Increment score attribute of the current PacmanBoard by 10.
6.2. Pac-Man eats fruit : fruit disappear and score increment by 100
When Pac-Man moves onto the coordinate of a fruit, the fruit will be considered eaten. No modification on board is necessary here as it is done by update of Pac-Man coordinate. Increment score attribute of the current PacmanBoard by 100.
6.3. Pac-Man eats LARGE dots : large dots disappear and score increment by 50
When Pac-Man moves onto the coordinate of a large dot, the dot will be considered eaten. No modification on board is necessary here as it is done by update of Pac-Man coordinate. Increment score attribute of the current PacmanBoard by 50. Set value of "S" to 200. Refer to Backend 5.2, 5.3, and 7.1 for more use cases related to this one.
6.4. Pac-Man eats QUICKSILVER CHARM : Quicksilver CHARM disappears
When Pac-Man moves onto the coordinate of Quicksilver charm, the charm will be considered eaten. Refer to Backend 7.2 for more use cases related to this one.

7. Pac-Man ghost consumables interaction:

7.1. Pac-Man eats LARGE dots : the ghosts turn DARK BLUE and then start FLASHING for a short period of time
After Pac-Man eats a large dot, all the ghosts' update strategy will be changed to "away from Pac-Man." The integers that represents ghost on the board will be replaced by one that represents a frightened ghost, so frontend will render dark blue/flashing ghosts. After a short period time (a few state updates), all the ghosts' update strategy will be changes to one that is not "away from Pac-Man." The integers that represents the frightened ghost on the board will be replaced by one that represents a normal ghost. Refer to Backend 5.2, 5.3, and 6.3 for more use cases related to this one.
7.2. Pac-Man eats Quicksilver charm : Pac-Man becomes untouchable by ghost

Endpoints (HTTPS)

Table View

URL	Method	Description	Request Parameters	Response Parameters
/{boardId}/start	GET	Start the game	NONE	NONE
/{boardId}/board	GET	Get board data	NONE	score: int,level: int,board: int[][]
/{boardId}/reset	GET	Reset the game	NONE	NONE
/{boardId}/change/{direction}	GET	Change direction of Pac-Man	direction : string	NONE
/{boardId}/setGhost	Post	Set the number of ghosts	ghostNum : int	NONE
/init	GET	Initialize the game board	NONE	boardId: int

Swagger View

- Starts the game

```
"/{boardId}/start": {
    "get": {
    "parameters": [
        {
        "name": "boardId",
        "description": "the id of the game board",
        "type": "string",
        "required": true,
        "in": "path"
        }
    ],
    "responses": {
        "200": {
        "description": "Start the game"
        },
        "400": {
        "description": "game already started"
        }
    }
    }
}
}
```

- Gets the most up-to-date (position & status wise) game board

```
"/{boardId}/board": {
    "get": {
```

```
"get": {
"consumes": [
    "application/json"
],
"produces": [
    "application/json"
],
"parameters": [
    {
    "name": "boardId",
    "in": "path",
    "description": "the id of the game board",
    "required": true,
    "type": "string"
    }
],
"responses": {
    "200": {
    "description": "Successfully got board data",
    "schema": {
        "type": "object",
        "properties": {
        "score": {
            "type": "integer",
            "format": "int32",
            "example": 10
        },
        "level": {
            "type": "integer",
            "format": "int32",
            "example": 2
        },
        "life": {
            "type": "integer",
            "format": "int32",
            "example": 3
        },
        "board": {
            "type": "array",
            "items": {
            "type": "array",
            "items": {
                "type": "integer"
            }
            },
            "example": [
            [
                1,
                2,
                3
            ],
            [
                4,
                5,
                1
            ],
            [
                2,
                3,
                4
            ]
            ]
        }
        }
    }
    }
```

```
    }
    }
}
```

- Resets the game after GAME OVER before user starts a new game

```
"/{boardId}/reset": {
    "get": {
    "parameters": [
        {
        "name": "boardId",
        "in": "path",
        "description": "the id of the game board",
        "required": true,
        "type": "string"
        }
    ],
    "responses": {
        "200": {
        "description": "Reset the game"
        }
    }
    }
}
```

- Makes Pac-Man moves in user specified direction

```
"/{boardId}/change/{direction}": {
    "get": {
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ],
    "parameters": [
        {
        "name": "direction",
        "in": "path",
        "description": "The direction to change the pacman. Should be one of {up, down, right, left}",
        "required": true,
        "type": "string"
        },
        {
        "name": "boardId",
        "in": "path",
        "description": "the id of the game board",
        "required": true,
        "type": "string"
        }
    ],
    "responses": {
        "200": {
        "description": "Successfully changed the direction for the user"
        },
        "400": {
        "description": "Input direction is not valid. direction should be one of {up, down, right, left}"
        }
    }
    }
}
```

- Set the number of ghosts

```
"/{boardId}/setGhost/": {
    "post": {
    "consumes": [
        "application/json"
    ],
    "parameters": [
        {
        "name": "ghostNum",
        "description": "number of ghost in game, should be 3 - 8",
        "in": "body",
        "schema": {
            "type": "object",
            "properties": {
            "ghostNum": {
                "type": "integer",
                "example": 3
            }
            }
        }
        },
        {
        "name": "boardId",
        "description": "the id of the game board",
        "type": "string",
        "required": true,
        "in": "path"
        }
    ],
    "responses": {
        "200": {
        "description": "successfully set ghost num"
        },
        "400": {
        "description": "ghost num invalid, should be 3 - 8"
        },
        "404": {
        "description": "board id not found"
        }
    }
    }
}
}
```


- Landing. Initialize a game board

```
"/init": {
    "get":  {
    "produces": ["application/json"],
    "responses": {
        "200": {
        "description": "successfully created a new game board",
        "schema": {
            "type": "object",
            "properties": {
            "boardId": {
                "type": "integer",
                "format": "int32",
                "example": 10
            }
            }
        }
        }
    }
    }
}
```