Josefa Osorio
CSE40793

Chip's Challenge Final Deliverable

Previous Features:

CellBehavior:
+ getImageView()
+ getX()
+ getY()
+ canEnter()
+ isBug()
+ hasWon()

Grid:
- Grid grid
- CellBehavior cellGrid[][]
- int dimensions
- int totalChips
- int chipsCollected
+ getGrid()
+ getSize()
+ setCell()
+ getCell()
+ addToTotalChips()
+ addToChipsCollected()
+ getIfAllChipsCollected()
+ getChipsLeft()
+ cellSet()
+ clear()

Chip *implements CellBehavior*:
- ImageView tileIV
- Point position
- String path
+ getPath

Bug *implements CellBehavior*:
- ImageView tileIV
- Point position

ChipItem *implements CellBehavior*:
- ImageView tileIV
- Point position

Key *implements CellBehavior*:
- ImageView tileIV
- Point position
- String path
- String color
+ getPath()

KeyWall *implements CellBehavior*:
- ImageView tileIV
- Point position
- String path
- String color
+ getPath()

New Features:

Game *extends Application*:
- int win_width
- int win_length
- AnchorPane root
- Scene scene
- Grid grid
- int level
- Player player
- State state
- Score scorekeeper
- Enemy enemy
+ reset()
+ startGame()

IVController:
- int imageSize
- int scalingFactor
- Image tileImage
- ImageView tileImageView
+ getTileIV()

LevelController:
- Grid grid
- AnchorPane root
- CellFactory cellFactory
- int dim
+ getLevelGrid()
+ drawLevelOne()
+ drawLevelTwo()

Player:
- int dimensions
- Player player
- Point position
- ArrayList<String> keys
- State state
- Chip chipCell
- Grid grid
- boolean canMove
+ getInstance()
+ getPosition()
+ getImageView()
+ getChipCell()
+ getState()
+ move()
+ getPosition()
+ takeKey()
+ useKey()
+ hasKey()
+ canMove()
+ changeState()

Score:
- int win_width
- int win_length
- GraphicsContext gc
- Canvas canvas
- AnchorPane root
- int fontSize
- int x
- int y
+ isDead()
+ updateText()
+ nextLevel()
+ wonGame()
+ update()

Enum **State**:
- PLAYING
- DEAD
- WON
- String str
+ getString()

Inside CellsUtils:

Enemy:
- Enemy enemy
- Point position
- Grid grid
- int level
- Bug bug
- boolean canMove
- boolean dirMoving
+ getInstance()
+ getPosition()
+ getImageView()
+ getBug()
+ canMove()
+ move()

BlankTile *implements CellBehavior*:
- ImageView tileIV
- Point position

CellFactory:
+ addCell()

Enum **CellType:**
- CHIP
- CHIPITEM
- BUG
- KEYWALL
- WALLTILE
- KEY
- PORTAL
- PORTALGATE
- BLANKTILE

Enum **Orientation:**
- DOWN
- UP
- LEFT
- RIGHT
- String str
+ getString()

Portal *implements CellBehavior*:
- ImageView tileIV
- Point position

PortalGate *implements CellBehavior*:
- ImageView tileIV
- Point position

WallTile *implements CellBehavior*:
- ImageView tileIV
- Point position

UML Diagram:

The UML Diagram doesn't include the features above to save space. Please assume the features are inside of the boxes.

** UML Diagram is added as a separate PDF available in my GitHub repository under the filename: josorio2_final_deliverable_uml_diagram.pdf **

Discussion:

My original design included three separate factories for the different items that could be added to the screen: chip items, keys, and key walls. I then had a different Behavior Interface for Chip, the player, and then a Bug Factory for the bugs. Beginning to write the code for this program, I found that I would need more clases, but less factories. Instead of having four different factories, I could have one overarching Cell Behavior Interface and a factory that created different types of cells/characters that implemented the behavior shown in the Cell Behavior Interface.

Another change that I noticed that I needed to implement was changing how to implement the Grid. I originally had the Grid creating the Scene and holding the root, but I decided that the Grid would hold CellBehavior objects and set work as a middle-man between the game, the players, and the different cell types. I created a Game class that extended Application so that reset and started the game. I also had it use an AnimationTimer so if an enemy (in this case a Bug) was part of the game, it would display like an animation rather than only moving when a key was pressed or not at all. The Game Class is where the main function is.

The two controller classes I had were an ImageView Controller and a LevelController. I created the ImageView Controller class because creating an ImageView object was repeated in each Cell object and this would simplify the process. I created the LevelController Class to consolidate the action of drawing the levels into one class based on what level it was. I used a State class to determine where the player was in their playing of the game: playing, dead, or won. Finally I had a Score class that output the text onto the screen with what level the player was on and how many chips they had left to collect. I also included message when the player died or won.

If I could have done something differently, I would have tried implementing the State method for the different states that the player was in. I did a rough implementation of it by just creating an enum class named State, but I believe a class that implemented it correctly would have added greater benefits to the program. I also would have tried implementing a Composite-type design pattern for the Enemies so that the enemy would have an array list of different types of enemies, not just bugs, so that it could keep track of where the enemies are and where they have to go. To extend that, in a future level, the Observer pattern could have been used with the enemy objects to follow the player to make it more difficult.

Design Patterns:

Pattern Name: Factory Method	
Class Names	**Role in Pattern**
CellFactory	Factory
Blank Tile, Bug, Chip, ChipItem, Key, KeyWall, Orientation, Portal, PortalGate, WallTile	Items Created by Factory
Purpose: Many different objects implementing CellBehavior needed to be created and this would reduce the amount of functions needed to be implemented by Grid and could consolidate the setting of the cells to one Factory Class containing two addCell functions: one for regular Cells and one for Cells with color attributes.	

Pattern Name: Strategy	
Class Names	**Role in Pattern**
CellBehavior	Strategy Interface
Blank Tile, Bug, Chip, ChipItem, Key, KeyWall, Orientation, Portal, PortalGate, WallTile	Strategy Classes implementing the interface
Purpose: Many different objects needed to have similar functionalities with only a couple classes with a few additions so by creating this Strategy interfact, it allowed each of the Cell types to implement all of the functions in the CellBehavior Interface.	

Pattern Name: Singleton	
Class Names	**Role in Pattern**
Grid	Single Object
Purpose: The Grid class has its constructor as private and has a static instance of itself. This allows for only the Grid class to create instances of itself and the only way to access the object would be through the Grid Class, in this case by calling the getGrid() method. This ensures that only one Grid object is created at a time.	