TCP2201 Key Collector Project

Trimester 1, 2017/2018

by Invisible Wing

Team Leader:
Goh Kun Shun
017-285 9398
kunshun225@gmail.com

Team members:
Christopher Too Wei Bin
010-291 3385
christophertwoweibin@gmail.com

Ng Jing Keong
017-212 5595
james0523njk@gmail.com

Ong Koon Hua
010-280 2826
koonhua310@gmail.com

Instructions

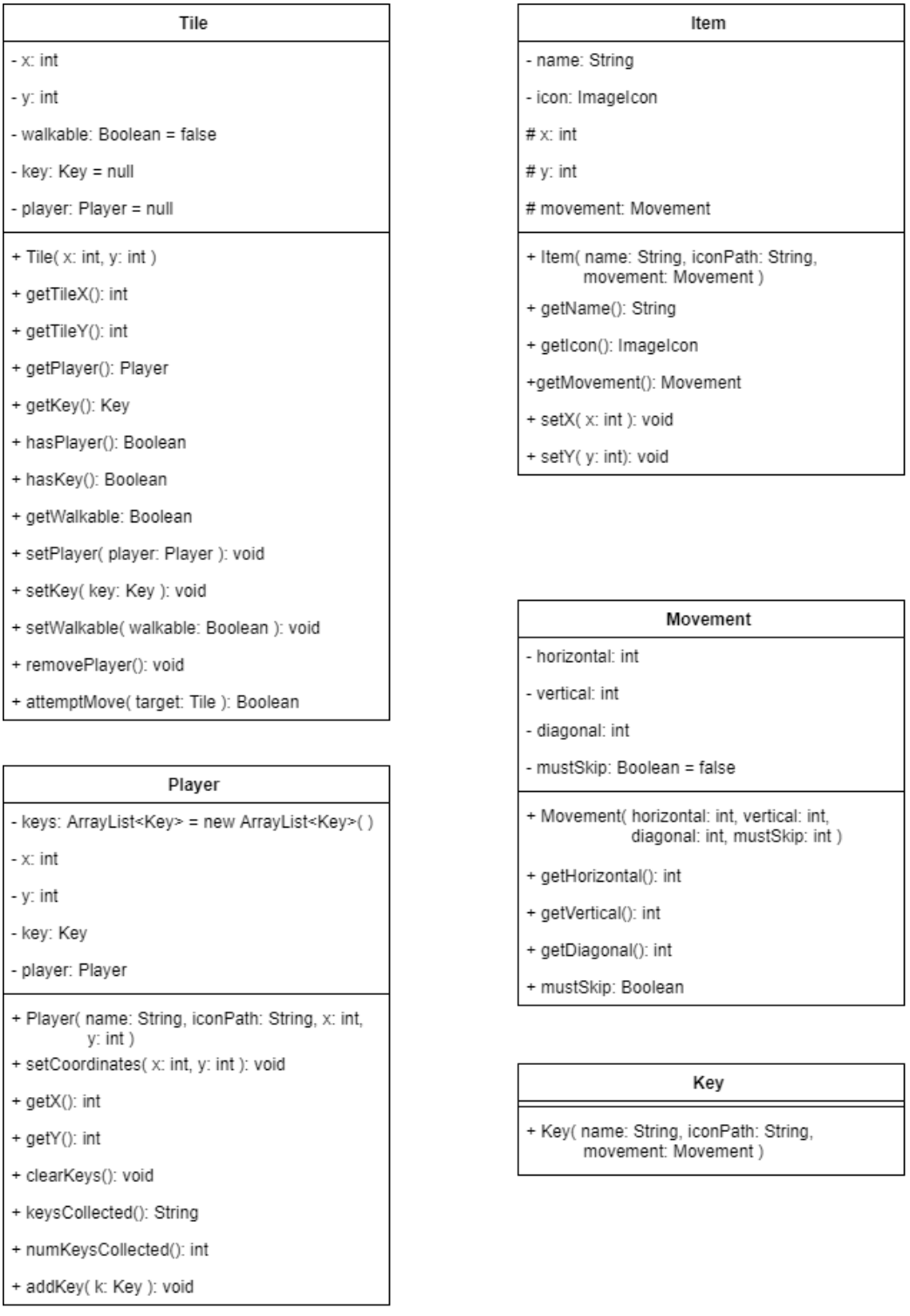
To run the program from the command line, unzipped the zip file into any folder. Set the directory of the command line to the directory where the files are extracted to. The extracted files should include the following:

- KeyCollectorGame.java
- Tile.java
- Item.java
- Player.java
- Key.java
- Movement.java
- icons folder containing:
    - 1.gif
    - 2.gif
    - 3.gif
    - 4.gif
    - 5.gif
    - a.gif
    - b.gif
    - c.gif
    - d.gif
    - e.gif

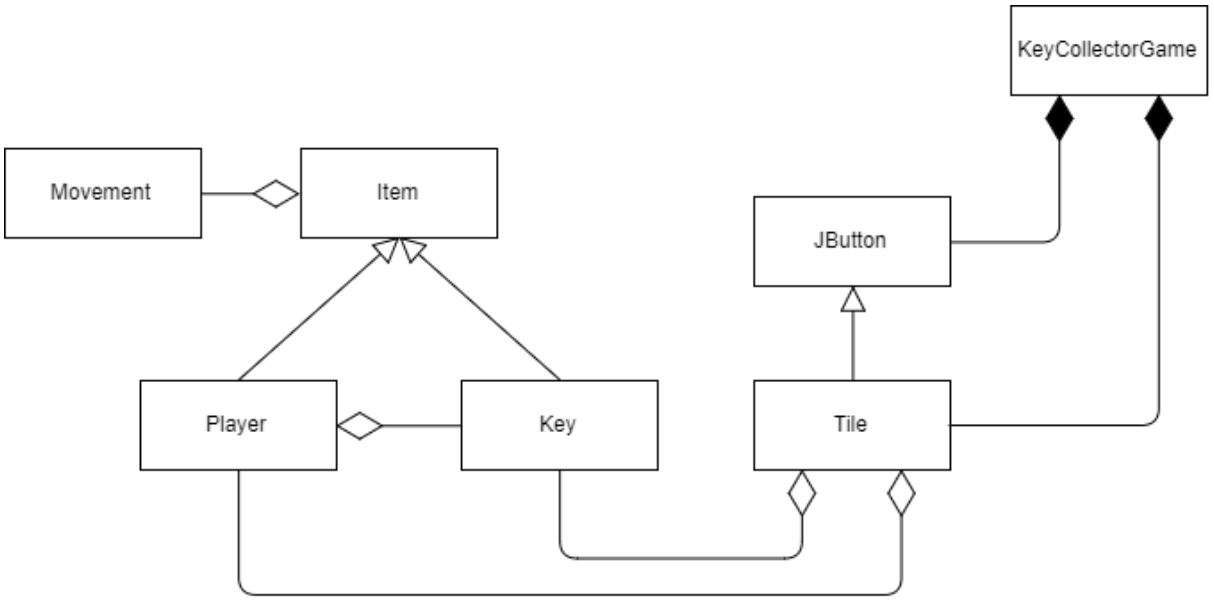
In the command line, type in *"javac KeyCollectorGame.java"* and press enter to compile the .java files. Notice that the .class files will be created for each corresponding .java file. Then, type in *"java KeyCollectorGame"*. A windows will popup and the game starts!

UML Class Diagram

Below are the UML class diagram for the classes we created, **Tile**, **Item**, **Player**, **Key**, **Movement** and **KeyCollectorGame**.

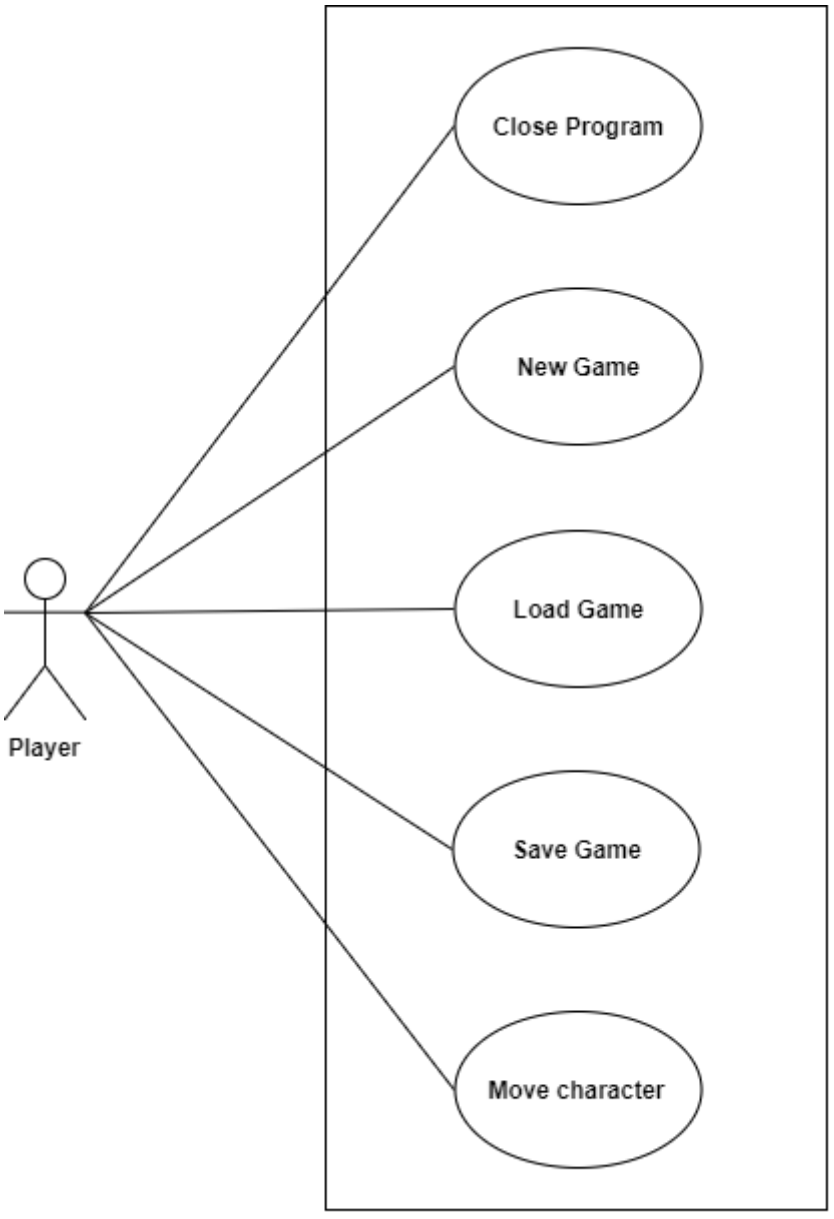



KeyCollectorGame
~ tiles: Tile [ ] [ ] = new Tile [9] [9]
~ players: Player [ ] = Player [4]
~ keys: Key[ ] = new Key [6]
~ current_player: Player
~ current_tile: Tile
~ round: int = 0
~ gameOver: Boolean = false
~ gameStatus: String
~ statusPanel: JPanel = new JPanel(new FlowLayout ( ));
~ statusLabel: javax.swing.JLabel = new javax.swing.JLabel(" ")
~ actionPanel: JPanel = new JPanel(new BorderLayout ( ))
~ menuPanel: JPanel = new JPanel(new FlowLayout ( ))
~ actionLabel: javax.swing.JLabel = new javax.swing.JLabel(" ")
+ main: void
+ KeyCollectorGame()
- initialize(): void
- randomPlaceKeys(): void
- checkWalkable(): void
- resetWalkable(): void
- updateGameStatus(): void
- updateActionLabel(s: String): void
- newGame(): void
- clearBoard(): void
- loadGame(): void
- saveGame(): void
+ actionPerformed(ae: ActionEvent): void

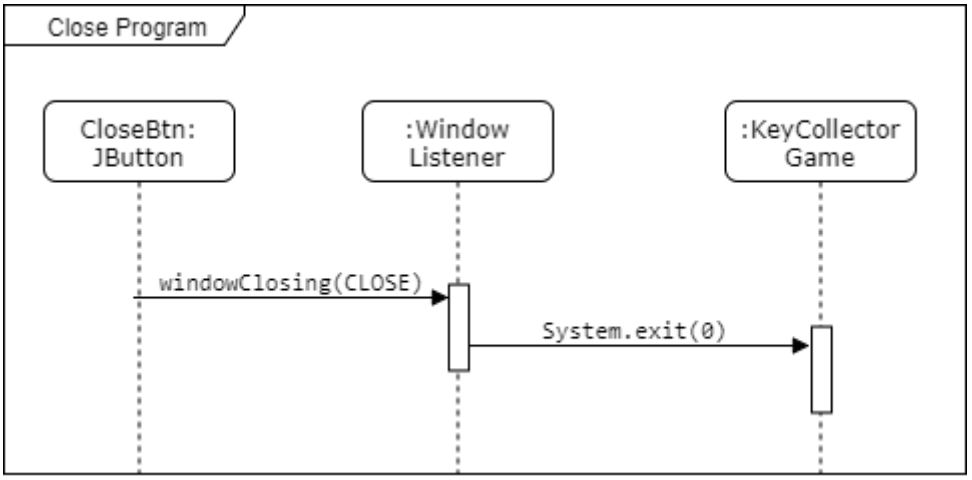


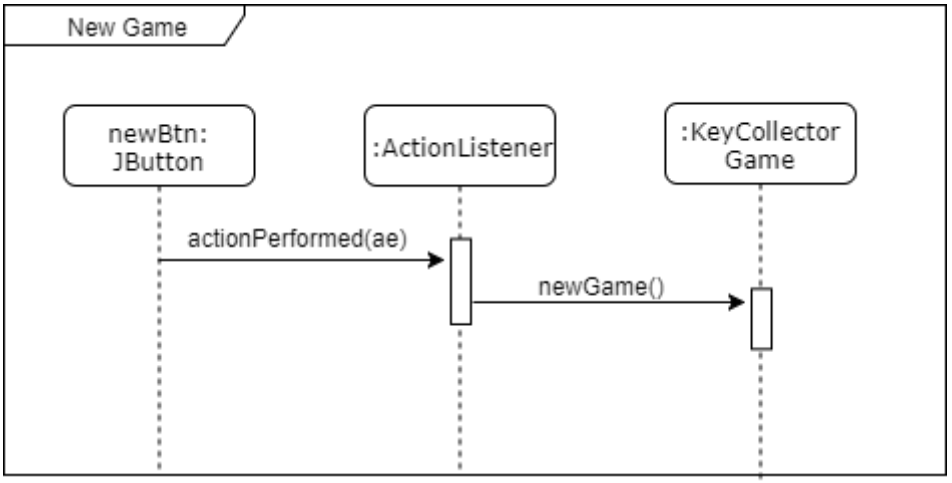
The figure above shows how the classes are related to each other. Every **Item** has a **Movement**. A **Player** and **Key** is an **Item**. A **Tile**, which inherits from the **JButton** can have a **Key** and a **Player**. A **Player** can have none to many **Key**s. The **KeyCollectorGame** is composed of **JButton**s and **Tile**s.

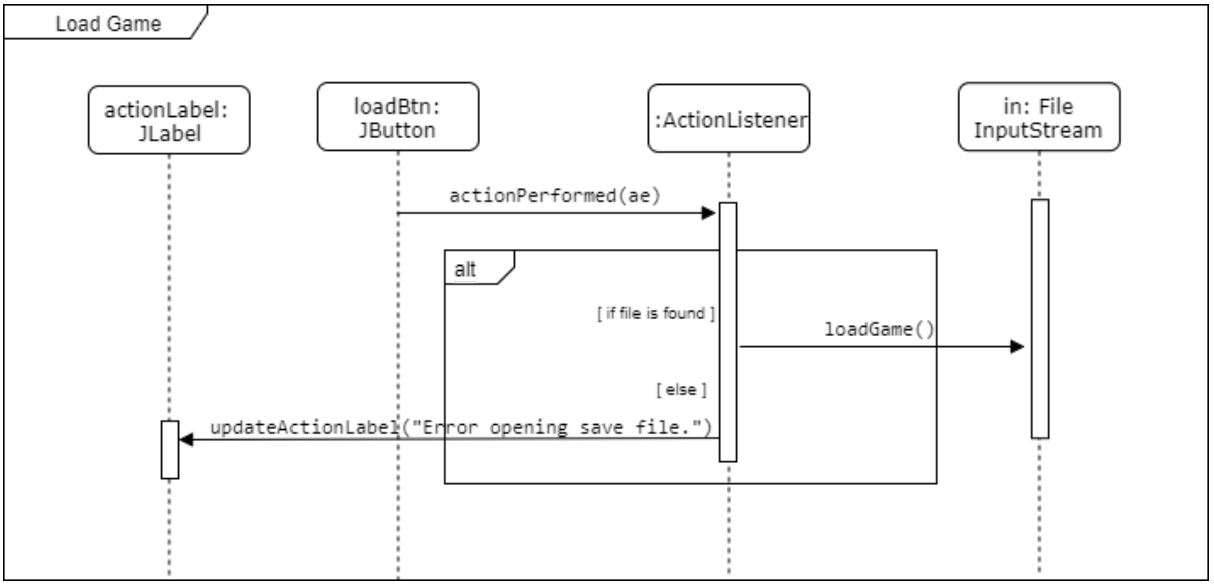
Use Case Diagram

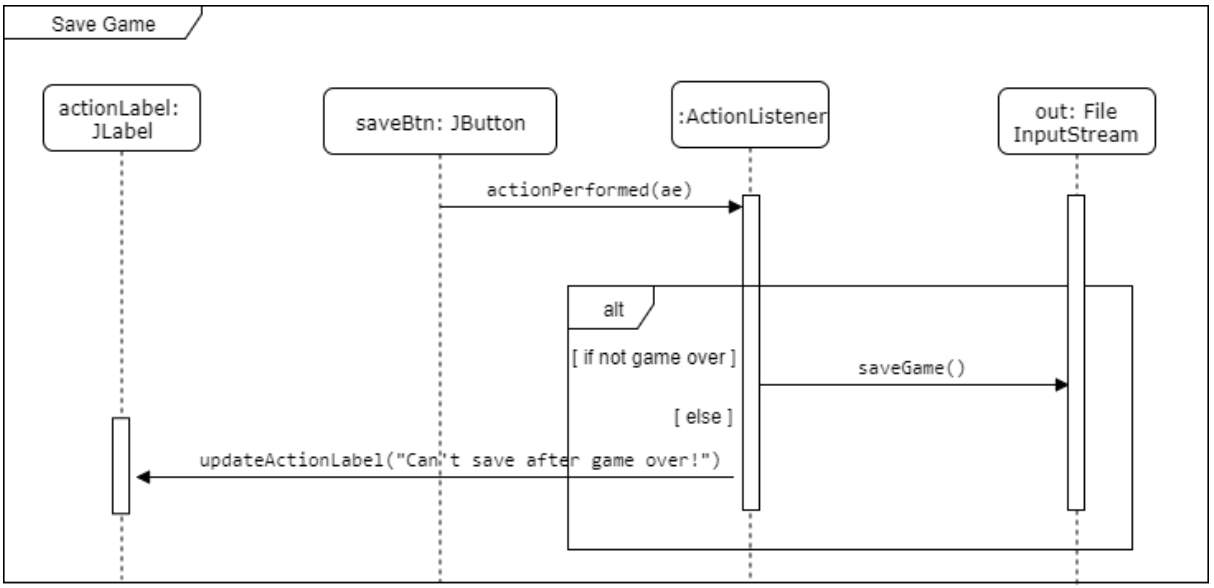


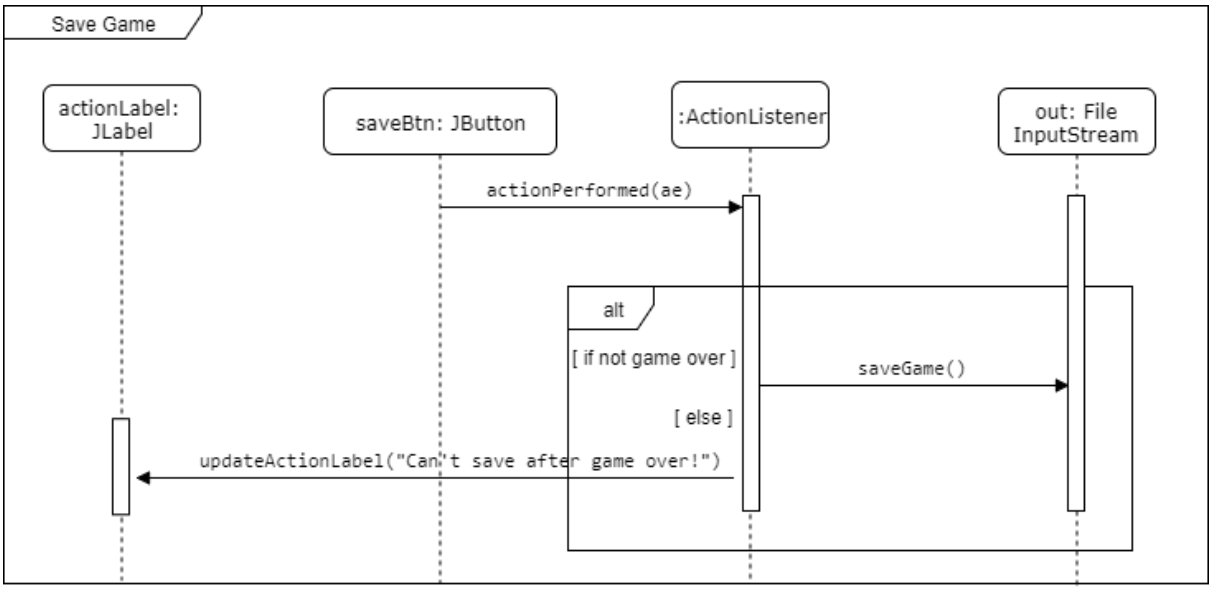
Sequence Diagrams








Save Game
actionLabel: JLabel
saveBtn: JButton
:ActionListener
out: File InputStream
actionPerformed(ae)
alt
[if not game over]
saveGame()
[else]
updateActionLabel("Can't save after game over!")


Save Game
actionLabel: JLabel
saveBtn: JButton
:ActionListener
out: File InputStream
actionPerformed(ae)
alt
[if not game over]
saveGame()
[else]
updateActionLabel("Can't save after game over!")