XBlast

Software Architecture

XBlast

McGill University
ECSE 321 – Winter 2014

Authors:

David Liu
Yike Liu
Eddy Lu
Xuzhi Shu
YaHan Yang
Christopher Reny

March 2nd, 2014

Table of Contents

System Overview

GameSystem

The design's core is the GameSystem class. This class creates a JFrame and adds to it a canvas. It then adds KeyEventListeners to the canvas. GameSystem also implements runnable and makes sure the tick method is ran 30 times a second maximum.

Three main components of GameSystem are the Game class, Menu class, and Story class. Each of them has the tick and render method, so GameSystem can easily determine what to do based on what state it's currently in based on the enumeration SystemState. For example, if the SystemState == Game, the tick and render method in GameSystem will only call the tick and render methods in the instance of the Game class.

KeyEvents

KeyEvents are handled in a very similar way. When a KeyEvent is detected, it will simply be passed to the instance of Game, Menu, or Story depending on current SystemState. The KeyEvent will then be handled.
game's settings. In addition, the user has the ability to end the game at will.

Game

The Game class is active when the player is engaging in active moving around and placing bombs. It contains a set of storable Data, GameData. This is a Classe that could be serialized and stored into a file. Thus when the player saves the game, the system will first update GameData and then saves it to file. When the player loads, the gets the instance of GameData from the file and then set it to the instance of GameData in the instance of Game.

Controller

Another important component of Game is the Controller class. The Controller class stores, creates, updates, and renders all GameObjects. GameObjects are mostly things that could be seen during playing time which include the player, enemies, bombs, fire, bricks and other stuff.

When a new level is started, the Game class will simply assign a new Controller and then create GameObjects from scratch. This setup is done in the LevelLoader class.

Menu

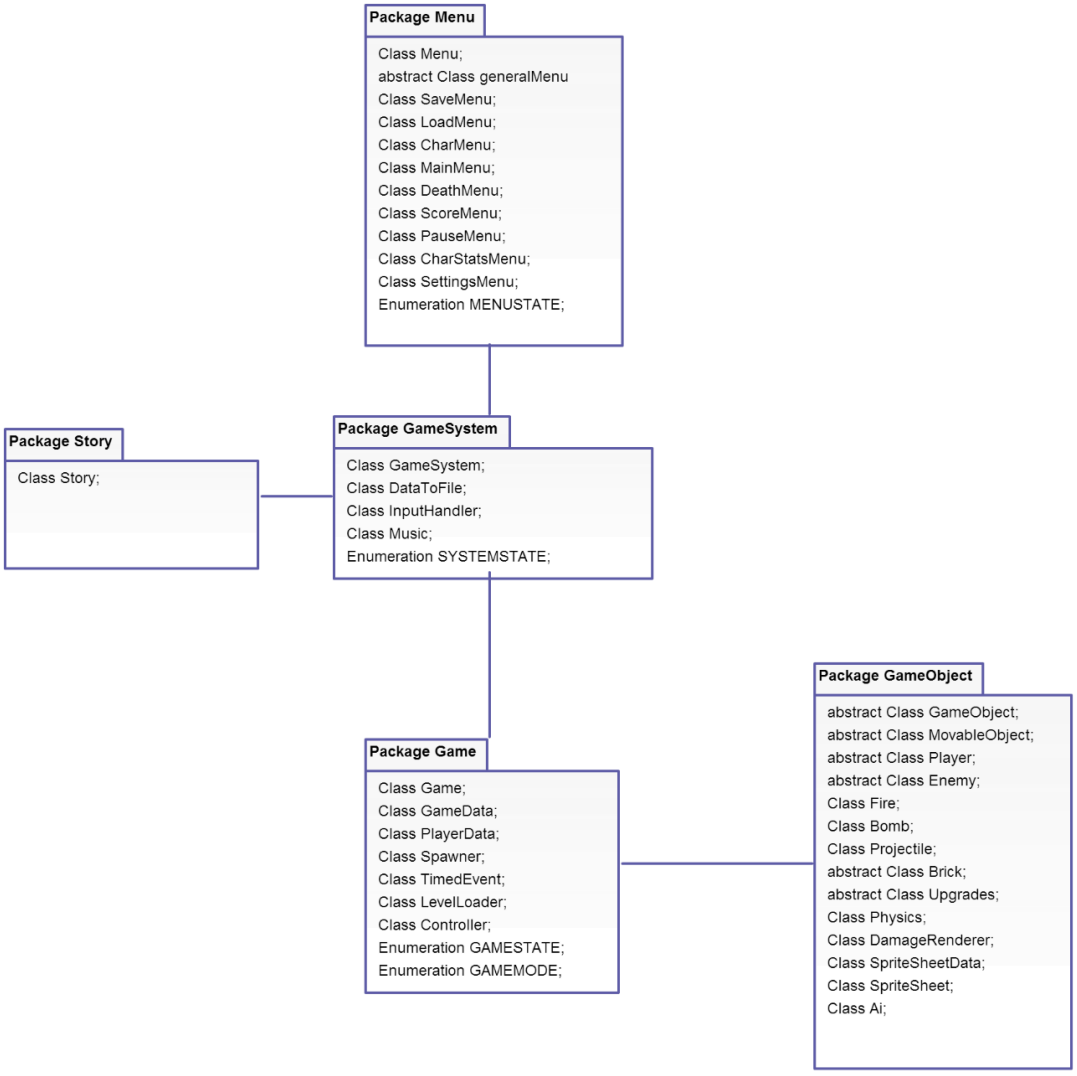
The Menu class is fairly simple. It is composed of a lot of classes, each is set to render a specific menu. The Menu class will determine which menu to render depending on an enumeration.

Story

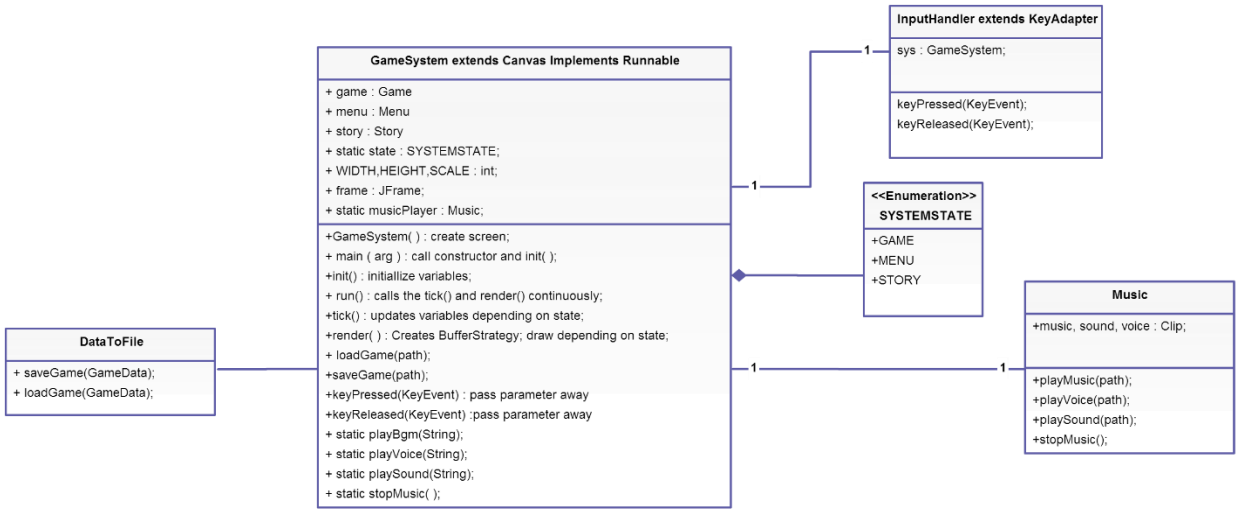
The Story class is used to create a storyline. It will read from a text file. It has a lineNum variable which determines how many lines to render onto the screen. All the read lines from the file are stored in a String Array of size x, were x is the maximum allowed number of lines to be on the screen at once. The Story class also provides a method to trigger specific actions. If some specific lines are read from the file, it will do certain stuff such as changing the background picture.

Views

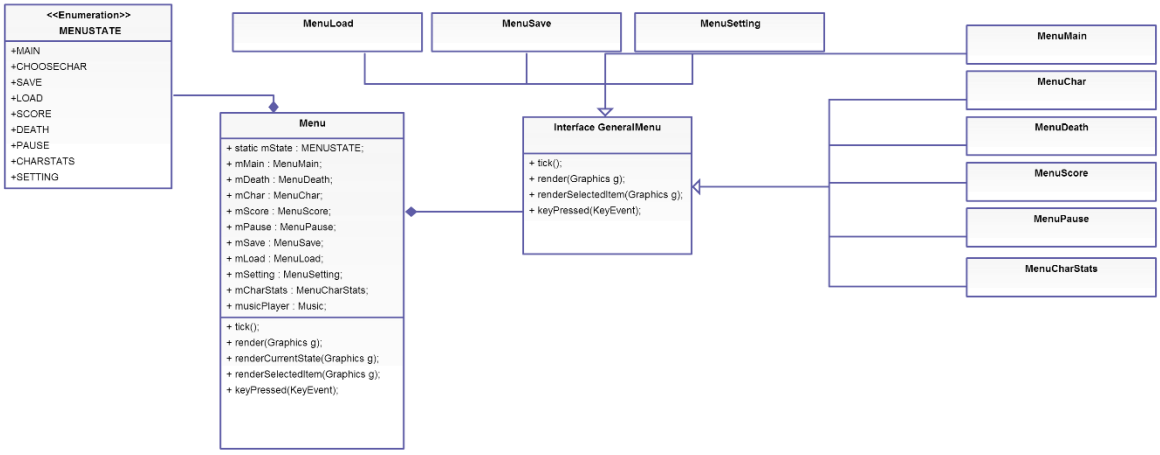
Overview



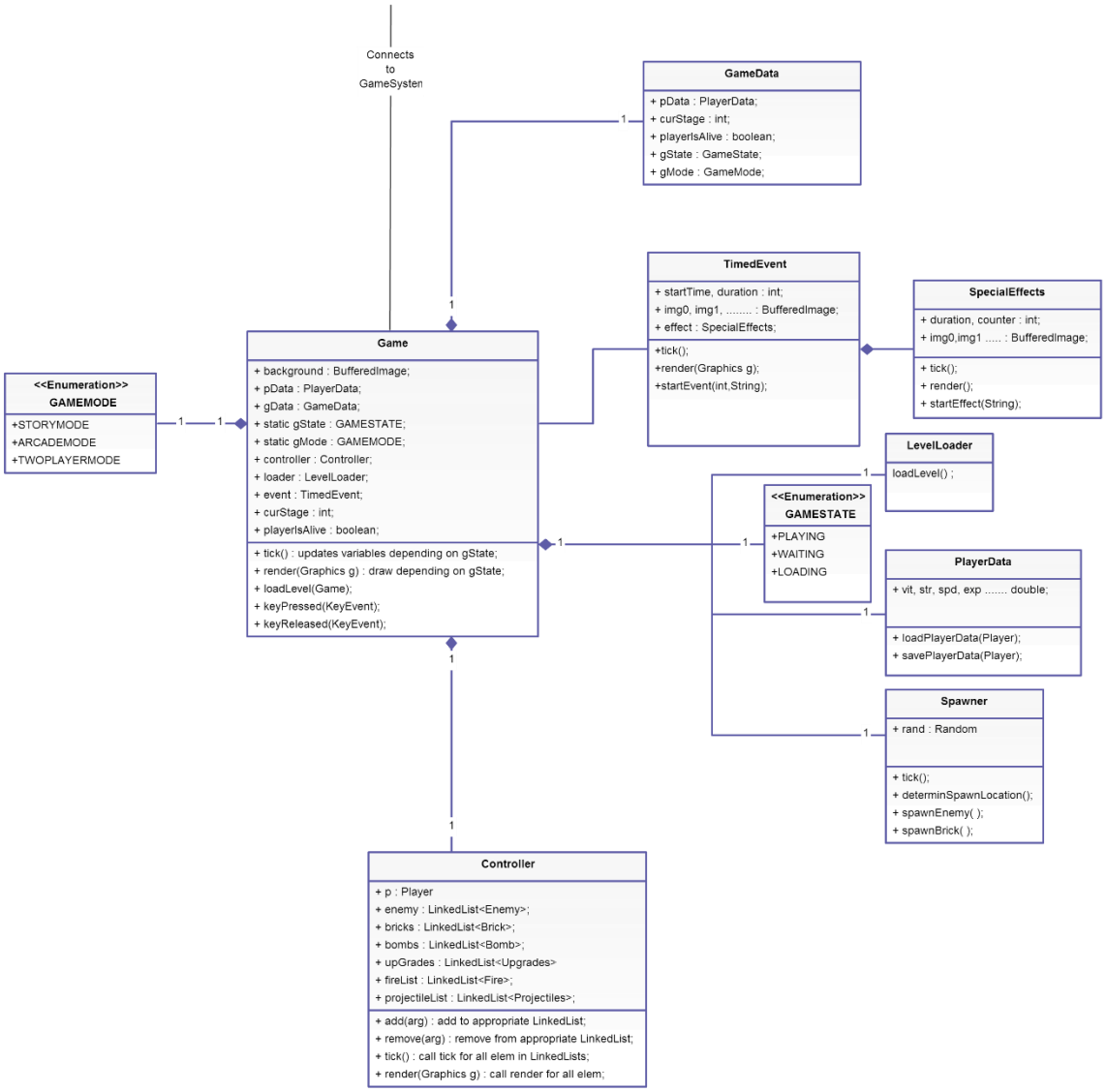
GameSystem



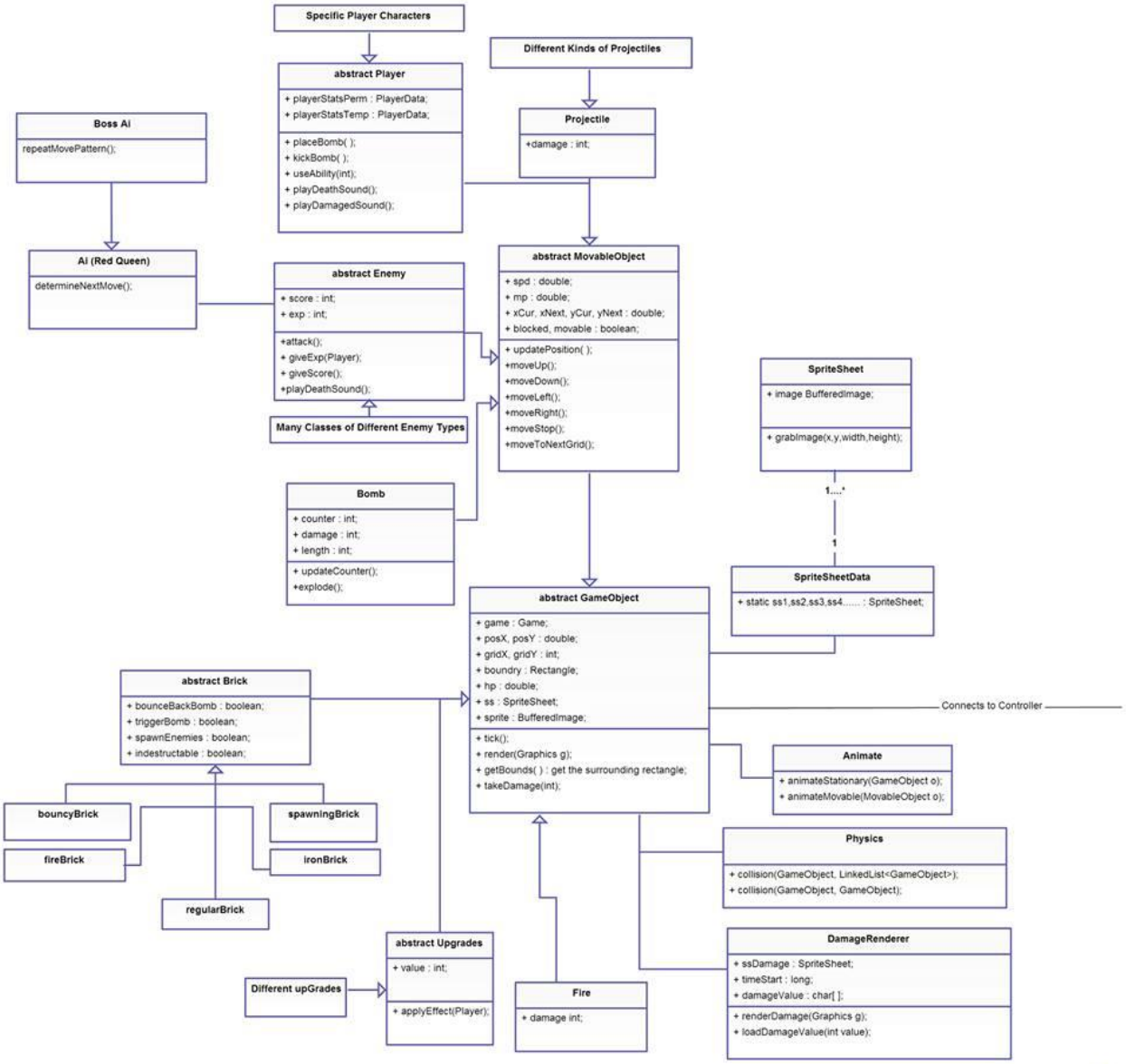
Menu



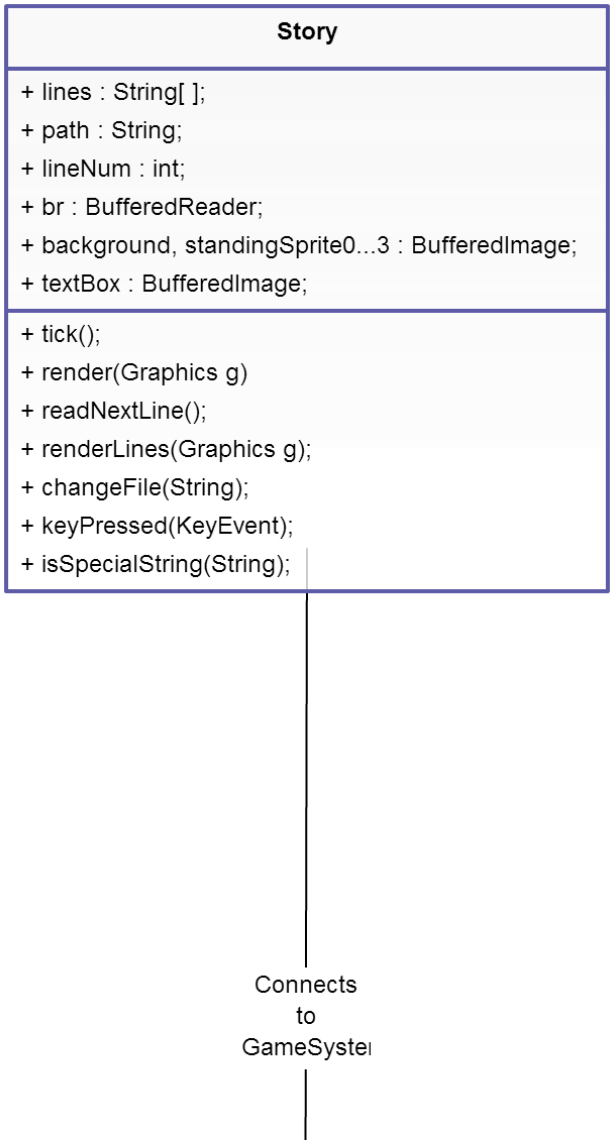
Game


Connects to GameSystem
GameData
+ pData : PlayerData;
+ curStage : int;
+ playerIsAlive : boolean;
+ gState : GameState;
+ gMode : GameMode;
TimedEvent
+ startTime, duration : int;
+ img0, img1, ... : BufferedImage;
+ effect : SpecialEffects;
+tick();
+render(Graphics g);
+startEvent(int,String);
SpecialEffects
+ duration, counter : int;
+ img0,img1 ..... : BufferedImage;
+ tick();
+ render();
+ startEffect(String);
<<Enumeration>>
GAMEMODE
+STORYMODE
+ARCADEMODE
+TWOPLAYERMODE
Game
+ background : BufferedImage;
+ pData : PlayerData;
+ gData : GameData;
+ static gState : GAMESTATE;
+ static gMode : GAMEMODE;
+ controller : Controller;
+ loader : LevelLoader;
+ event : TimedEvent;
+ curStage : int;
+ playerIsAlive : boolean;
+ tick() : updates variables depending on gState;
+ render(Graphics g) : draw depending on gState;
+ loadLevel(Game);
+ keyPressed(KeyEvent);
+ keyReleased(KeyEvent);
LevelLoader
loadLevel() ;
<<Enumeration>>
GAMESTATE
+PLAYING
+WAITING
+LOADING
PlayerData
+ vit, str, spd, exp ..... double;
+ loadPlayerData(Player);
+ savePlayerData(Player);
Spawner
+ rand : Random
+ tick();
+ determinSpawnLocation();
+ spawnEnemy( );
+ spawnBrick( );
Controller
+ p : Player
+ enemy : LinkedList<Enemy>;
+ bricks : LinkedList<Brick>;
+ bombs : LinkedList<Bomb>;
+ upGrades : LinkedList<Upgrades>
+ fireList : LinkedList<Fire>;
+ projectileList : LinkedList<Projectiles>;
+ add(arg) : add to appropriate LinkedList;
+ remove(arg) : remove from appropriate LinkedList;
+ tick() : call tick for all elem in LinkedLists;
+ render(Graphics g) : call render for all elem;

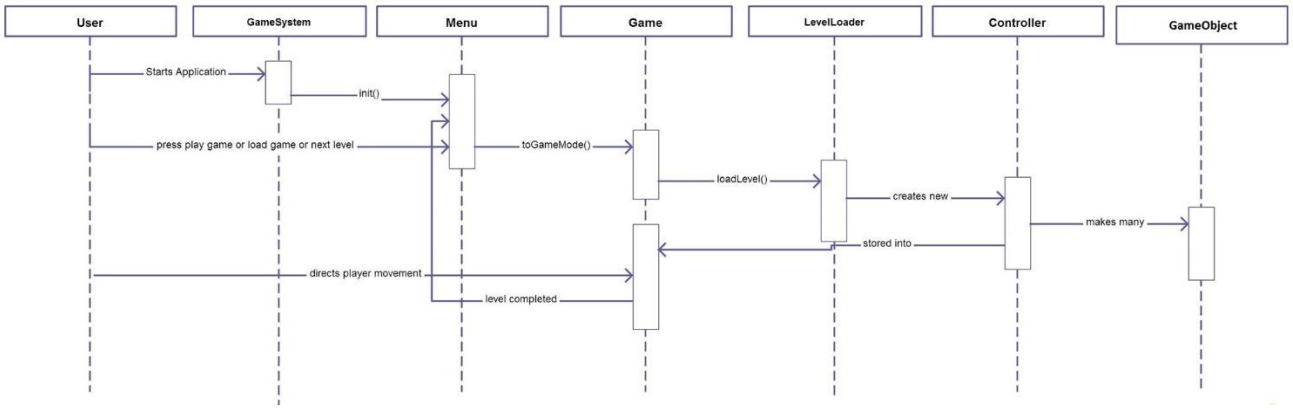
GameObject


Specific Player Characters
Different Kinds of Projectiles
abstract Player
+ playerStatsPerm : PlayerData;
+ playerStatsTemp : PlayerData;
+ placeBomb( );
+ kickBomb( );
+ useAbility(int);
+ playDeathSound();
+ playDamagedSound();
Projectile
+damage : int;
Boss Ai
repeatMovePattern();
Ai (Red Queen)
determineNextMove();
abstract Enemy
+ score : int;
+ exp : int;
+attack();
+ giveExp(Player);
+ giveScore();
+playDeathSound();
Many Classes of Different Enemy Types
abstract MovableObject
+ spd : double;
+ mp : double;
+ xCur, xNext, yCur, yNext : double;
+ blocked, movable : boolean;
+ updatePosition( );
+moveUp();
+moveDown();
+moveLeft();
+moveRight();
+moveStop();
+moveToNextGrid();
SpriteSheet
+ image BufferedImage;
+ grabImage(x,y,width,height);
1....*
1
SpriteSheetData
+ static ss1,ss2,ss3,ss4..... : SpriteSheet;
Bomb
+ counter : int;
+ damage : int;
+ length : int;
+ updateCounter();
+explode();
abstract GameObject
+ game : Game;
+ posX, posY : double;
+ gridX, gridY : int;
+ boundry : Rectangle;
+ hp : double;
+ ss : SpriteSheet;
+ sprite : BufferedImage;
+ tick();
+ render(Graphics g);
+ getBounds( ) : get the surrounding rectangle;
+ takeDamage(int);
Connects to Controller
abstract Brick
+ bounceBackBomb : boolean;
+ triggerBomb : boolean;
+ spawnEnemies : boolean;
+ indestructable : boolean;
Animate
+ animateStationary(GameObject o);
+ animateMovable(MovableObject o);
bouncyBrick
fireBrick
spawningBrick
ironBrick
regularBrick
Physics
+ collision(GameObject, LinkedList<GameObject>);
+ collision(GameObject, GameObject);
abstract Upgrades
+ value : int;
+ applyEffect(Player);
Different upGrades
Fire
+ damage int;
DamageRenderer
+ ssDamage : SpriteSheet;
+ timeStart : long;
+ damageValue : char[ ];
+ renderDamage(Graphics g);
+ loadDamageValue(int value);

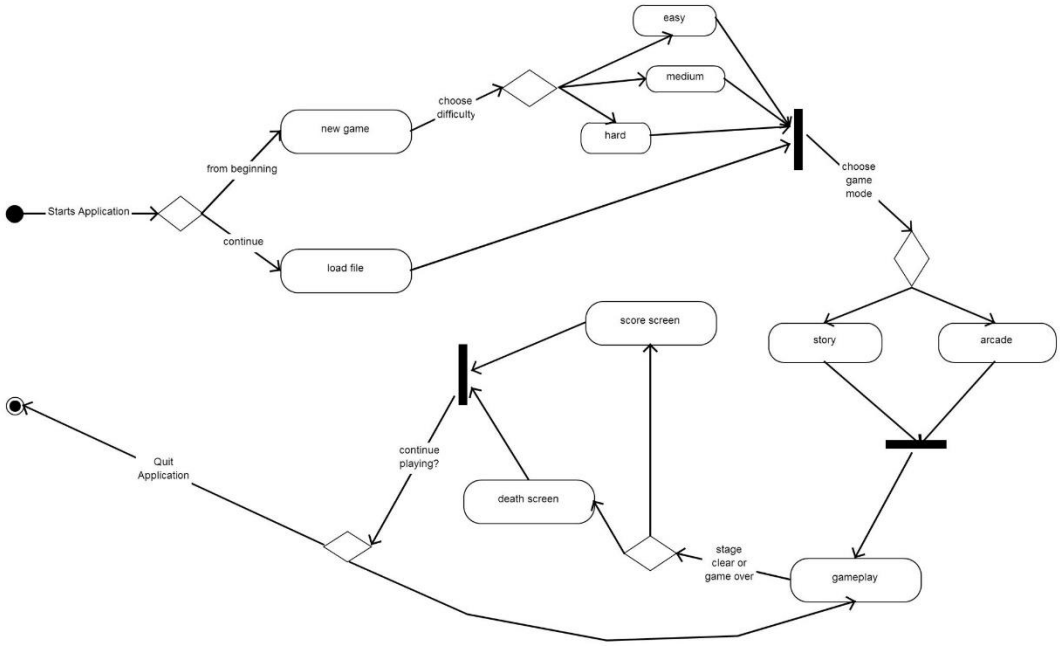
Story


Story
+ lines : String[ ];
+ path : String;
+ lineNum : int;
+ br : BufferedReader;
+ background, standingSprite0...3 : BufferedImage;
+ textBox : BufferedImage;
+ tick();
+ render(Graphics g)
+ readNextLine();
+ renderLines(Graphics g);
+ changeFile(String);
+ keyPressed(KeyEvent);
+ isSpecialString(String);
Connects
to
GameSyste

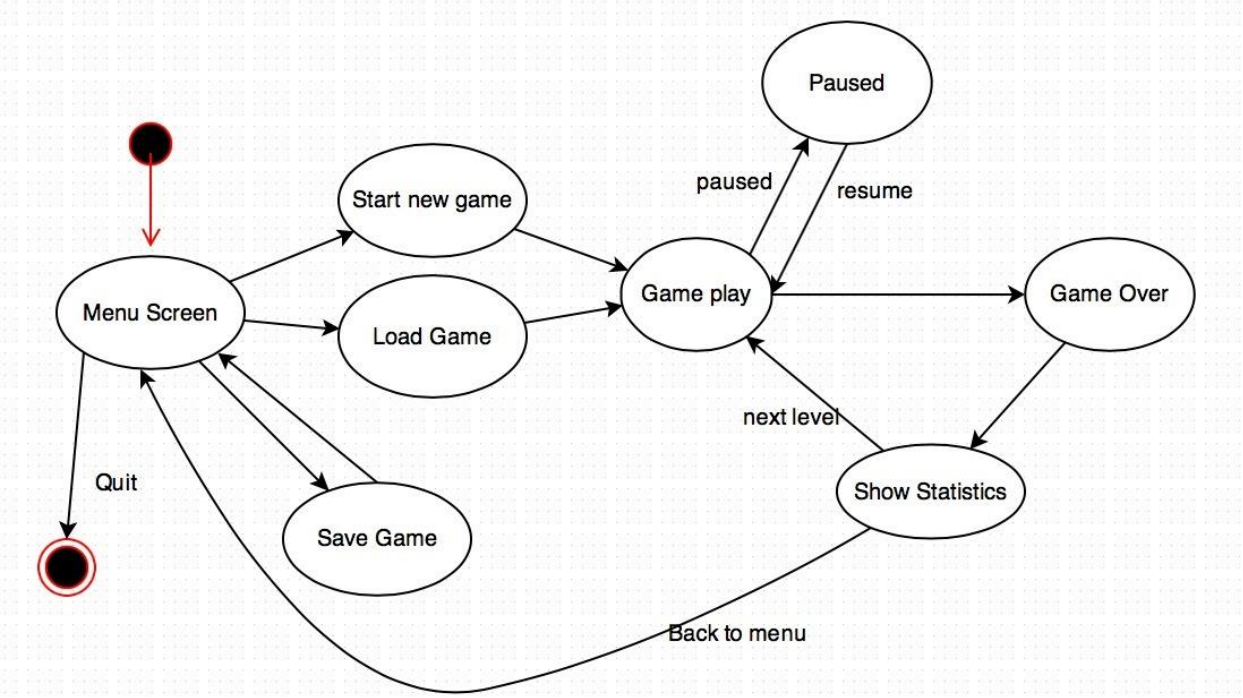
Sequence Diagram



Activity Diagram



State Diagram



Software Systems and Modules

Package GameSystem

Class GameSystem **extends Canvas implements runnable**

Purpose

this class serves as the starting point of the application. It creates a game loop that runs at maximum 30 times a second. It creates the window and assigns a BufferStrategy so stuff can be drawn. It also handles user input.

Methods

Void main(String args[])

Creates an instance of GameSystem and calls init() and start();

GameSystem()

Constructor; set size of the canvas; creates a JFrame and adds the canvas to it.

Void Init()

Assigns values to all the Global variable in the GameSystem class.

Void start()

Start a thread of the GameSystem.

Void run()

Calls tick() at most 30 times a second. Call render() continuously.

Void tick()

Checks the current SYSTEMSTATE; if state is GAME then call game.tick(); if state is MENU then call menu.tick(); if state is STORY then call story.tick(); overall, tick() is used to update variables and check conditions periodically.

Void render()

Creates a BufferStrategy; renders either game, menu or story based on current SYSTEMSTATE; overall, render() will draw graphics onto the canvas, which is in the game window.

Void saveGame(String path)

Creates a save file at the indicated path.

Void loadGame(String path)

Loads the save file at the indicated path.

Void keyPressed(KeyEvent e)

Passes the KeyEvent down to game, menu, or story.

Void keyReleased(KeyEvent e)

Passes the KeyEvent down to game, menu, or story.

Void playBgm(String path)

Void playVoice(String path)

Play the file in the given path.

Void playSound(String path)

Play the file in the given path.

Void stopMusic()

Stops the current bgm.

Class DataToFile

Purpose

Allows the game to perform save and load options.

Methods

Void saveGame(GameData gData)

Serializes the given gData.

Void loadGame(GameData gData)

Sets gData to data read from file.

Class InputListener

Purpose

Allows detection of user inputs.

Methods

Void keyPressed(Keyevent e)

Pass e back to GameSystem.keyPressed and it will be handled later.

Void keyReleased(Keyevent e)

Pass e back to GameSystem.keyReleased and it will be handled later.

Class Music

Purpose

Contains 3 Clips: music, sound, and voice. Allows audio to play.

Methods

Void playBgm(String path)

Sets the music clip to the file at path and loops the clip.

Void playVoice(String path)

Sets the voice clip to the file and path and plays it once.

Void playSound(String path)

Sets the sound clip to the file and path and plays it once.

Void stopMusic()

Stops the music.

Package Game

Class Game

Purpose

Allows the player to play the game. Handles player inputs for controlling movement and actions; stores player information and game information, which could be written and load from file.

Methods

Void tick()

Does the following periodically:
If the player is dead, change to the death menu state.
If victory condition is achieved, change to score menu state.
Else call tick from the instance of Controller.

Void render(Graphics g)

Draws the background of the stage;
Calls the render method from the instance of the Controller.

Void loadLevel(Game game)

Loads the current level with the LevelLoader class.

Void keyPressed(KeyEvent e)

allows the player move around in the map, place bomb and use abilities.

Void keyReleased(KeyEvent e)

Similar to keyPressed.

Class GameData implements Serializable

Purpose

stores a bunch of double, int, and boolean variable that will need to be used by the save and load feature.

Methods

No methods.

Class PlayerData

implements Serializable

Purpose

Stores variables that is needed when creating a player. These variables will be stored into file with the save feature.

Methods

Void savePlayerData(Player p)

take the current data of the player type and update the corresponding field in the instance of PlayerData accordingly.

Void loadPlayerData(Player p)

take the data in this instance of PlayerData and set the player's variable to the corresponding values.

Class Spawner

Purpose

Create more enemies when the player is playing.

Methods

Void tick()

Create enemies when certain conditions are met.

Void determineSpawnLocation()

Determines where the enemies are created.

Void spawnEnemy()

Makes a new subclass of Enemy.

Void spawnBrick()

Makes a new subclass of Brick.

Class TimedEvent

Purpose

Allows some specific action to occur for only an given amount of time.

Methods

Void tick()

> Updates the timer.
> Handles variables that need updating for each specific event.
> This method does nothing when the timer is greater than the event duration.

Void render(Graphics g)

> Draws out graphics for each specific event if they are active.
> This method will do nothing when the timer is greater than the event duration.

Void startEvent(int duration,String key)

> Resets the timer and sets the duration of the event. Also sets the key, which allows the class to identify what to update and draw.

Class LevelLoader

Purpose

This class can create a controller and create GameObjects with the controller. Thus, making a level of the game.

Methods

Constructor(Game game)

Sets a global variable game to the given instance of Game.

Void loadLevel()

Creates a new Controller;
Creates different GameObjects in the controller depending on what level the game is at.
Finally, set game.controller to the newly created controller.

Class Controller

Purpose

Stores different kinds of GameObject into different LinkedLists for both organization and collision handling. Tick and render GameObjects. Creates and removes GameObjects from LinkedLists.

Methods

Void tick()

Calls tick() for all GameObjects stored in the LinkedLists.

Void render(Graphics g)

Calls render(Graphics g) for all GameObjects stored in the LinkedLists.

Void add(GameObject o)

Add the input to an appropriate LinkedList depending on what specific type of GameObject it is.

Void remove(GameObject o)

Remove the input from an an appropriate LinkedList depending on what specific type of GameObject it is.

Package GameObject

Abstract Class GameObject

Purpose

A GameObject is a visual component that could be placed rendered onto the canvas. It may or may not move. GameObjects interact with the player, which is also a GameObject, through collisions. All GameObjects have a set of position variables that define were they are place and a Rectangle that define its boundry. All GameObjects are also associated with a SpriteSheet, which is a set of images, and a BufferedImage, which is what is drawn onto screen.

Methods

Constructor(int xGrid, int yGrid, Game game)

The input is the grid position of the GameObject. For example (1,1) means the upper left corner, and (1,3) is simply 2 grids down from (1,1).
The constructor creates the game object at the given location. Also converting the grid position into absolute positions that are used to render the image.
It also stores the input game variable into a global Game variable.

Void tick()

Does nothing for now because not all GameObject need to tick(). It will be overwritten in more specific subclasses.

Void render(Graphics g)

Renders the current image at current position.

Void getBounds()

Returns the rectangle at the player's location, of the size of the player's image's width and height.

Void takeDamage(int dValue)

Updates the GameObject's current hp according to dValue.

Abstract class MovableObject extends GameObject

Purpose

MovableObjects have methods that update their position variables. It also has many other parameters that determine how fast it can move, it's next position, and whether it's blocked by an obstacle. A MovableObject will also be animated when it moves.

Methods

Void tick()

First check if the MovableObject is blocked; if blocked do nothing.
If not blocked call updatePosition();

Void updatePosition()

update the position variables with respect to the velocity variables.

Void moveup()

Sets velY = -speed; velX = 0;

Void moveDown()

Sets velY = speed; velX=0;

Void moveLeft()

Sets velX = -speed; velY = 0;

Void moveRight()

Sets velX = speed; velY = 0;

Void moveStop()

Sets velX = velY = 0;

Void moveToNextGrid()

When the player stops moving, the MovableObject will automatically finish moving to the next grid.

Abstract Class Player extends MovableObject

Purpose

This is the MovableObeject that the player gets to control. It is abstract because there will be serveral playable characters. However, each character will have the same variable names and methods, but the values will be different.

Methods

Void placeBomb()

Places a bomb at the player's current xGridNearest and yGridNearest location, which is the closes grid to the player.

Void kickBomb()

If the player is standing on top of a bomb, make the bomb move at a certain speed at the direction the player is facing.

Void useAbility(int selected)

Use the selected ability.

Void playDamagedSound()

Play a sound effect to indicate that damage is received.

Void playDeathSound()

Makes the player character say some last words.

Abstract Enemy extends MovableObject

Purpose

Provides enemies that will try to kill the player. The player can kill enemies to get experience and level up to become stronger.

Methods

Void attack()

Creates a projectile or bomb MovableObject that will damage the player if collision is detected.

Void giveExp(Player p)

Gives experience equivalent to the exp value of the enemy to the inputed player.

Void giveScore()

Adds score to the player's current score.

Void playDeathSound()

Play some sound effect when destroyed.

Class Projectile extends MovableObject

Purpose

Is created when enemies attack and when player use certain abilities.

Methods

No new methods

Class Bomb extends MovableObjects

Purpose

A bomb is the player's primary way of dealing damage. A bomb object includes a timer which will update every time the tick method is called. A bomb is removed when the timer exceeds a certain value; in its place, fire of length x is created in 4 directions. A bomb could also be triggered when enemy projectiles collide with it, or when fire collides with it.
A bomb extends MovableObjects because it can be kicked to fly at enemies.

Methods

Void updateCounter()

Updates the counter variable.

Void explode()

Removes the bomb object form the LinkedList of bombs in the controller of the instance of the game and then create a bunch of fire objects instead.

Class Fire extends GameObject

Purpose

Fire is made when a bomb explodes. A fire object will be removed after 0.5seconds. It will deal damage if it comes into collision with a wall, player, or enemy.

Methods

No new methods

Abstract Class Brick extends GameObject

Purpose

A brick is used to block a grid. Nothing can move into a Brick GameObject. A brick can be destroyed if enough damage is dealt to it.

Methods

No new methods

Abstract Class Upgrades

Purpose

This defines the bonus drops that only the player can pick up to get power ups. There are many subclass of upgrades, but each will have a int value variable.

Methods

Void applyEffect(Player p)

Gives the player various benefits depending on what type of upgrade it is.

Class Physics

Purpose

Provides methods that detect if there is a collision.

Methods

Boolean collision(GameObject o, LinkedList<GameObject> list)

Return true if o is in collision with any element in list; false otherwise;

Boolean collision(GameObject a, GameObject b)

Return true if a is in collision with b; false otherwise.

Class DamageRenderer

Purpose

Whenever any damage is dealt, some images of integers will be drawn onto screen for a short duration to show how much damage is dealt.

Methods

loadDamageValue(int value)

loads the input damage value;

renderDamage(Graphics g)

draw nothing if duration is over.

Else draw the set of integer images corresponding to whatever value loadDamageValue loaded.

Class SpriteSheetData

Purpose

Stores all the SpriteSheets that are used by the program. All SpriteSheets are static.

Methods

No methods. The Class is used as storage only.

Class SpriteSheet

Purpose

A SpriteSheet is an image that's divided into several components, each component has the same width and height. Sub images can be grabbed from the SpriteSheet.

Methods

BufferedImage grabImage(x,y,width,height)

Returns the sub image at given coordinate.

Class Ai

Purpose

The Ai will control enemy MovableObject and try to kill the player. The Ai will determine when to call the moveUp, moveDown, moveLeft, moveRight, and moveStop methods.

Methods

Constructor(Enemy enemy)

Initializes instance

Void determineNextMove()

checks conditions and call methods depending on which are met.

Class Animate

Purpose

animate movable and immobile GameObjects by changing the image that's rendered onto screen.

Methods

static animateMovable(MovableObject o):

changes the MovableObject's image depending on it's movement direction.

static animateStationary(GameObject o):

loops through a sequence of a GameObject's image periodically, regardless of movement.

Package Menu

Class Menu

Purpose

The menu class provides the user with a GUI when the player is setting up the game, saving, loading, or viewing scores. The class creates many different types of GeneralMenu and determine which ones to show on screen depending on the current enumeration state.

Methods

Void tick()

Calls tick() from sub menus depending on the state.

Void render(Graphics g)

Calls renderCurrentState(Graphics g) and renderSelectedItem(Graphics g);

Void renderCurrentState(Graphics g)

Calls render(g) from the different sub menus depending on current state.

Void renderSelectedItem(Graphics g)

Calls renderSelectedItem(g) from the different sub menus depending on current state.

Void keyPressed(KeyEvent e)

Calls keyPressed(e) from the different sub menus depending on current state.

Interface GeneralMenu

Purpose

This is an interface implemented by all the sub menus.

Methods

Void tick()

Updates variables

Void render(Graphics g)

Draws stuff in the specific menu;

Void renderSelectedItem(Graphics g)

Render a new image over the option that the user have currently selected in a way that the user can tell what option is being selected.

Void keyPressed(KeyEvent e)

Handles the keyEvent finally.

Package Story

Class Story

Purpose

This class will allow the user to see a storyline. It reads lines from a file and store the lines into a String array of size x where x is the max number of lines that could be rendered onto the screen at ones.

Methods

Void tick()

Checks if background music is on.
If not on, turn it on.

Void render()

Renders background images, textboxes, String of text onto screen at appropriate places.

Void readNextLine()

Refreshes the array of Strings with the next line the BufferedReader reads from file. If the array is full, then store from index 0 again.

Void renderLines(Graphics g)

Only render the lines onto screen up until a int lineNum. When lineNum reaches above the size of the array that stores the Strings, lineNum is set back to 1;

Void changeFile(path)

load the another file and read from that one instead.

Boolean isSpecialString(String s)

If s is some pre-set string, this method will do some operation and return true. This is used so operations can be typed into the file, which this class will read from.

Analysis

Traceability Matrix

Class Corresponding Methods	Class InputListener	Class Music	Class AI	Class Animations animateMoveable	Class GameSystem saveGame	 loadGame	Class Menu
Inputs and Outputs							
1 Input Keys	x						
2 Player One Input Keys	x						
3 Player Two Input Keys	x						
4 Music		x					
5 Enemy AI			x				
6 Player Animations				x			
Stored Data							
1 Save Game					x		
2 Load Game						x	
Inialization/Shutdown							
1 Menu Input							x
2 Restart							x
3 Difficulty							x
4 Quit							x
Other Requirements							
1 Grid Display							
2 Stage Display							
3 Display Generation							
4 Beat Stage							x
5 Object Map							
6 Object Movement							
7 Object Bound Move							
8 Object Move Limit							
9 Health							
10 Bomb Placement							
11 Bomb Timer							
12 Bomb Capacity							
13 Fire Length							
14 Fire Obstacle							
15 Fire Health Decrease							
16 Bomb Disappear							
17 Random Item Appear							
18 Player Pick Up Items							
19 Item Benefit							
20 End Game							x
21 High Scores							

	Class	Class GameObject	Class Game	Class GameObject	Class Player Data	Class Player	Class Bomb		Class Fire	Class Upgrades
	Corresponding Methods	Constructor	Render	Constructor	savePlayerData		updateCounter	explode		applyEffect
Inputs and Outputs										
1	Input Keys									
2	Player One Input Keys									
3	Player Two Input Keys									
4	Music									
5	Enemy AI									
6	Player Animations									
Stored Data										
1	Save Game									
2	Load Game									
Inialization/Shutdown										
1	Menu Input									
2	Restart									
3	Difficulty									
4	Quit									
Other Requirements										
1	Grid Display	x								
2	Stage Display		x							
3	Display Generation			x						
4	Beat Stage									
5	Object Map			x						
6	Object Movement			x						
7	Object Bound Move			x						
8	Object Move Limit			x						
9	Health				x	x				
10	Bomb Placement					x				
11	Bomb Timer						x	x		
12	Bomb Capacity				x	x				
13	Fire Length								x	
14	Fire Obstacle								x	
15	Fire Health Decrease								x	
16	Bomb Disappear							x		
17	Random Item Appear									x
18	Player Pick Up Items									x
19	Item Benefit									x
20	End Game									
21	High Scores				x					

Quality Requirements

Performance, usability, and reliability requirement can only tested in the implementation phase.

Precision

1. The high score is stored as an integer.
   highscore is an integer field in PlayerData class.
2. The position (x, y) of movable game objects is stored as double.
   The position variables in MovableObjects are stored as doubles.

Reuse and Flexibility

The biggest example of reuse is the hierarchy formed with GameObject. Because lower level Classes automatically have access to higher level methods, it's very easy to add new Classes. Say for example I want to add new type of enemy. All I have to do is create a new Class, make it extend Enemy, and change some values such was which SpriteSheet is loads from and how much hp it has. All the movement methods will stay the same so nothing else has to be done.

Also, if I want to add a new SpriteSheet, I could just add it directly into the SpriteSheetData class and have GameObjects load it.

The software is maintainable as it is modular. All render and input operations first get passed into the base level and then handled. If later a new input need to be considered, it will just have to be added to the appropriate level in the structure.

Additional Extra Features

Player level up
Each play character will have a set of parameters that will be saved into file when the game saves. These data form the PlayerData Class. When a new player is created, it will load data form PlayerData.

Story mode
the story Class provides a storyline.

Various enemies and bosses
different enemy classes and bosses will extend the Enemy class.

Active abilities
special effects can be applied through the TimedEvent class, which activates some effect for a set amount of time.

Design Rationale

The bottom line of this design is that all user input go through a series of state checks and are handled at the very end. For example, when the user presses “z”, the GameSystem checks it’s state. If the state is Menu it passes it to the Menu class to handle. In the Menu class, when a KeyEvent is inputed it will check it’s state first. If it’s the main menu it will pass it to the keyPressed method in main menu. The main menu will then check which field is currently selected by the user. If the selection is on “start new game” then the keyEvent will lead to starting a new game.

Graphics rendering is done the same way except the parameter passed down is now a graphics variable.

Saving and loading works by serializing a Class that stores required fields in a game instance.

Workload Breakdown

David Liu	Class Ai
Yike Liu	Package Story
Eddy Lu	Package GameObject
Xuzhi Shu	Package GameSystem, Package Game
YaHan Yang	Package Menu
Christopher Reny	Package GameObject