Jumpy Jay

Relatório

Mestrado Integrado em Engenharia Informática e Computação

Laboratório de Programação Orientada por Objetos

Turma 2MIEIC01 - Grupo 14
201303882 João Miguel Fidalgo Esteves Nogueira 'up201304573@fe.up.pt'
201208067 Marta Milheiro Soeiro Nunes Lopes 'ei12106@fe.up.pt'

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

8 de Junho de 2015

1 Introdução

Este relatório tem como objetivo demonstrar como o nosso projeto funciona, projeto este, que é baseado no jogo *Jack N' Jill* disponível na *Play Store*.

O relatorio será constituído por um manual que vai explicar todas as funcionalidades do programa e o seu modo de utilização, a estrutura do código e a implementação do mesmo e também outras informações como os padrões de desenho utilizado e os mecanismos de jogo aplicados.

Jumpy Jay é um jogo de plataformas onde o utilizador vai controlar o *Robot Jay*. Este *robot* movimenta-se sozinho mantendo a mesma direção até embater numa parede ou num objeto que não seja letal, onde vai mudar de direção. O jogador apenas vai controlar o salto deste *robot* tentando apanhar os diamantes para aumentar o seu *score*, e apanhando a chave para poder abrir a saída e assim sair para terminar o nível. Existem vários tipos de perigos: os espinhos e a lava vão tirar apenas uma vida ao *robot*, sendo que este começa com 3 vidas no início de cada nível, os precipícios são os perigos onde se o *robot* cair, perde instantaneamente.

2 Manual de Utilização

2.1 Funcionalidades suportadas

O *Jumpy Jay* suporta as seguintes funcionalidades:

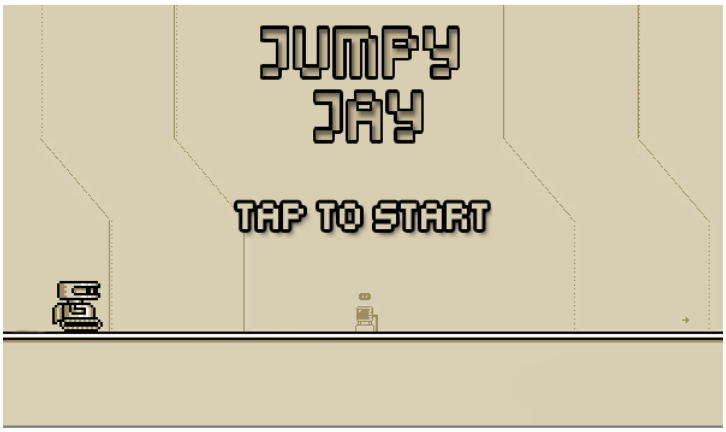
- Menu de Níveis: o utilizador vai poder escolher o nível que quer jogar, dependendo do seu progresso.
- Gravar/Ler o progresso do jogo.
- *Pause/Unpause*.
- Animações do *robot*: é possível visualizar à medida que ele se vai movimentado pelo mapa.
- Animações dos objetos no mapa e dos menus.
- Efeitos sonoros com *Mute/Unmute*.
- Música de fundo com *Mute/Unmute*.

2.2 Arranque do programa

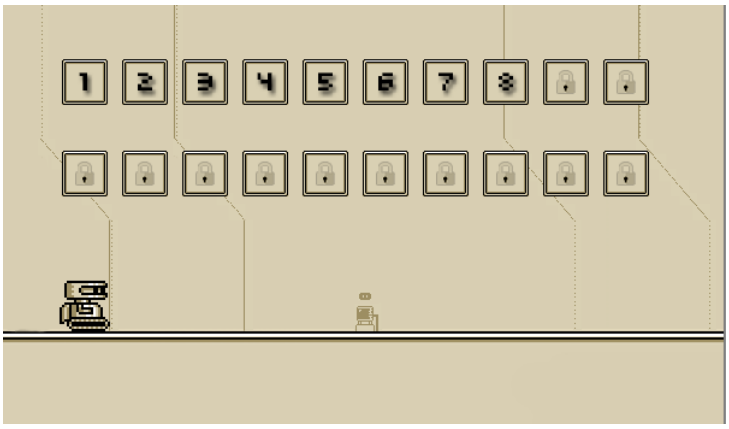
Para iniciar o jogo apenas necessita de abrir a aplicação *.jar* se estiver no computador, ou o *.apk* se estiver no sistema *Android*. O jogo começa com um ecrã de boas vindas onde o jogador dá um toque no ecrã para ir para o menu de níveis.

2.3 Modo de utilização

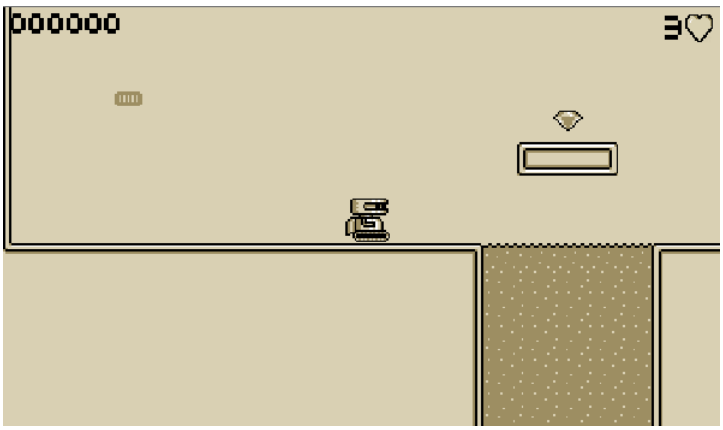
Menu Inicial: É apresentado um ecrã com algumas animações onde o jogador terá que dar um toque no ecrã para passar para o menu de níveis.



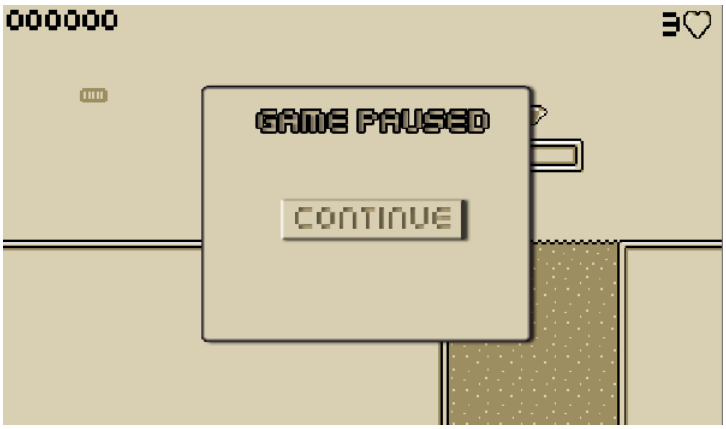
Menu Níveis: O jogador terá que escolher o nível em que quer jogar, dependendo do seu progresso no jogo. Para ter os níveis seguintes tem que passar os anteriores.



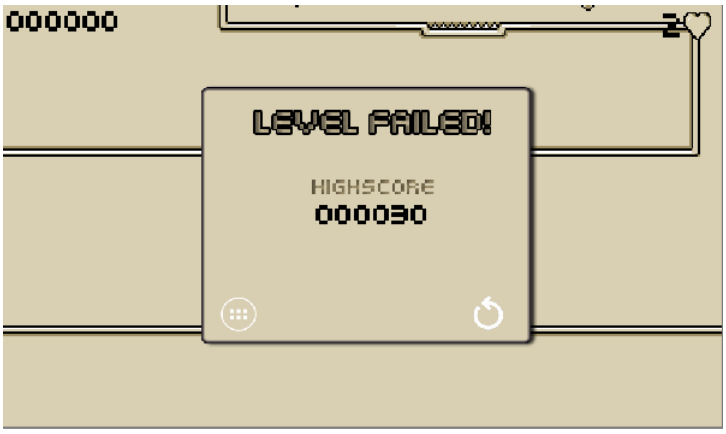
Jogo: Onde o utilizador vai jogar. O *Robot Jay* movimenta-se sozinho de um lado para o outro, mudando de direção quando bate numa parede. O jogador apenas controla quando é que o *robot* salta, dando um toque no ecrã.



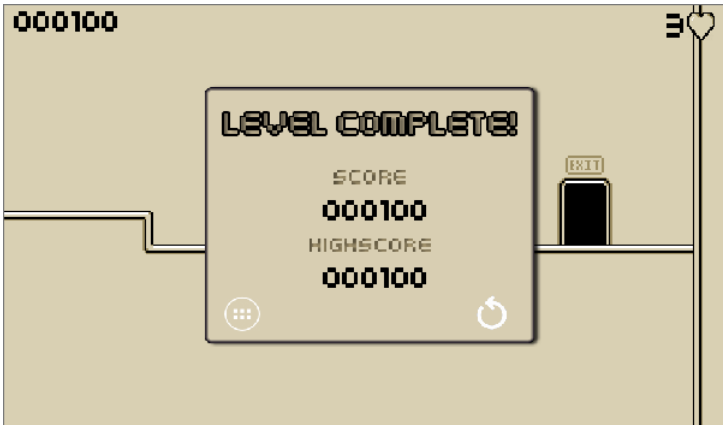
Pausa: Quando o jogador minimiza o jogo no telemóvel, ou clica fora do ecrã de jogo no computador, o jogo é pausado sendo que é possível ao utilizador regressar novamente ao ponto do jogo em que estava, clicando em *Continue*.



Perde: Quando o jogador perde, caindo num precipício ou caindo várias vezes nos espinhos ou na lava até ficar sem vidas é apresentado este ecrã. O jogador pode reiniciar o nível clicando no botão da direita, ou voltar ao menu de níveis clicando no botão da esquerda. É apresentado também o *highscore* daquele nível.



Ganha: Quando o jogador ganha, conseguindo apanhar a chave e chegando à saída, é apresentado o ecrã de vitória onde o jogador poderá reiniciar o nível clicando no botão da direita, ou voltar ao menu de níveis clicando no botão da esquerda. É apresentado também o seu *score* naquele nível e o *highscore* correspondente.

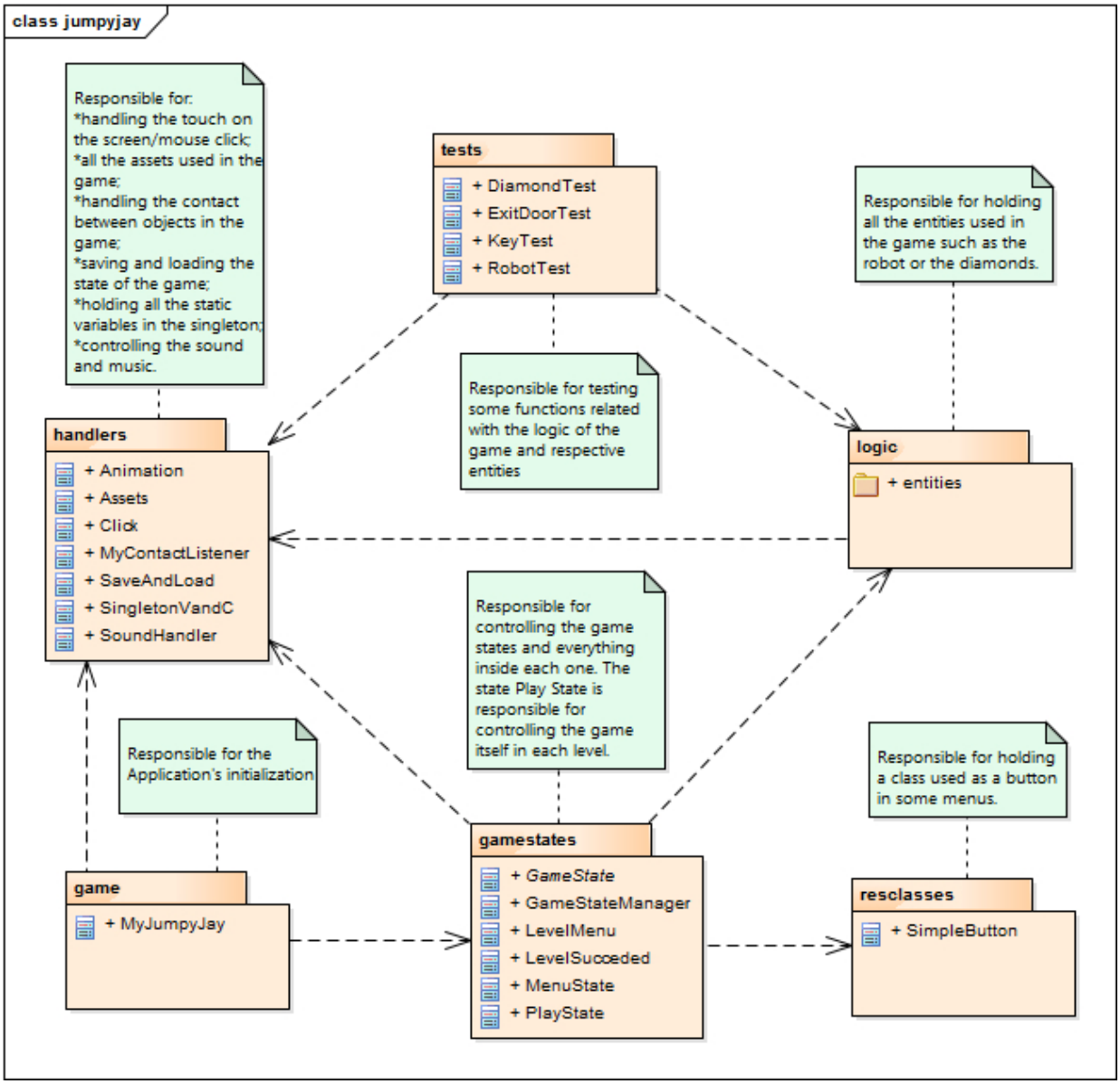


2.4 Formato dos ficheiros

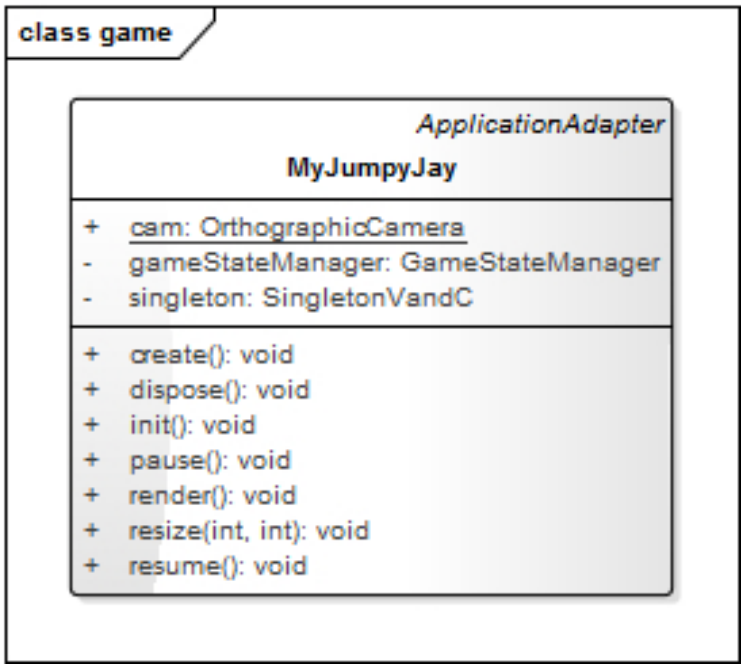
- **.jar** - executável em Java.
- **.apk** - ficheiro de instalação em *Android*.
- **.dat** - ficheiros de gravações

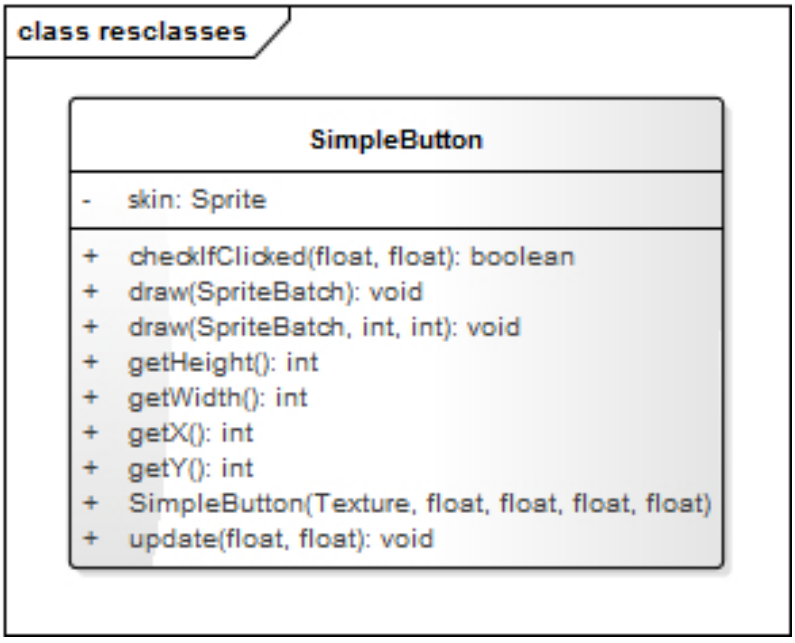
3 Conceção e Implementação

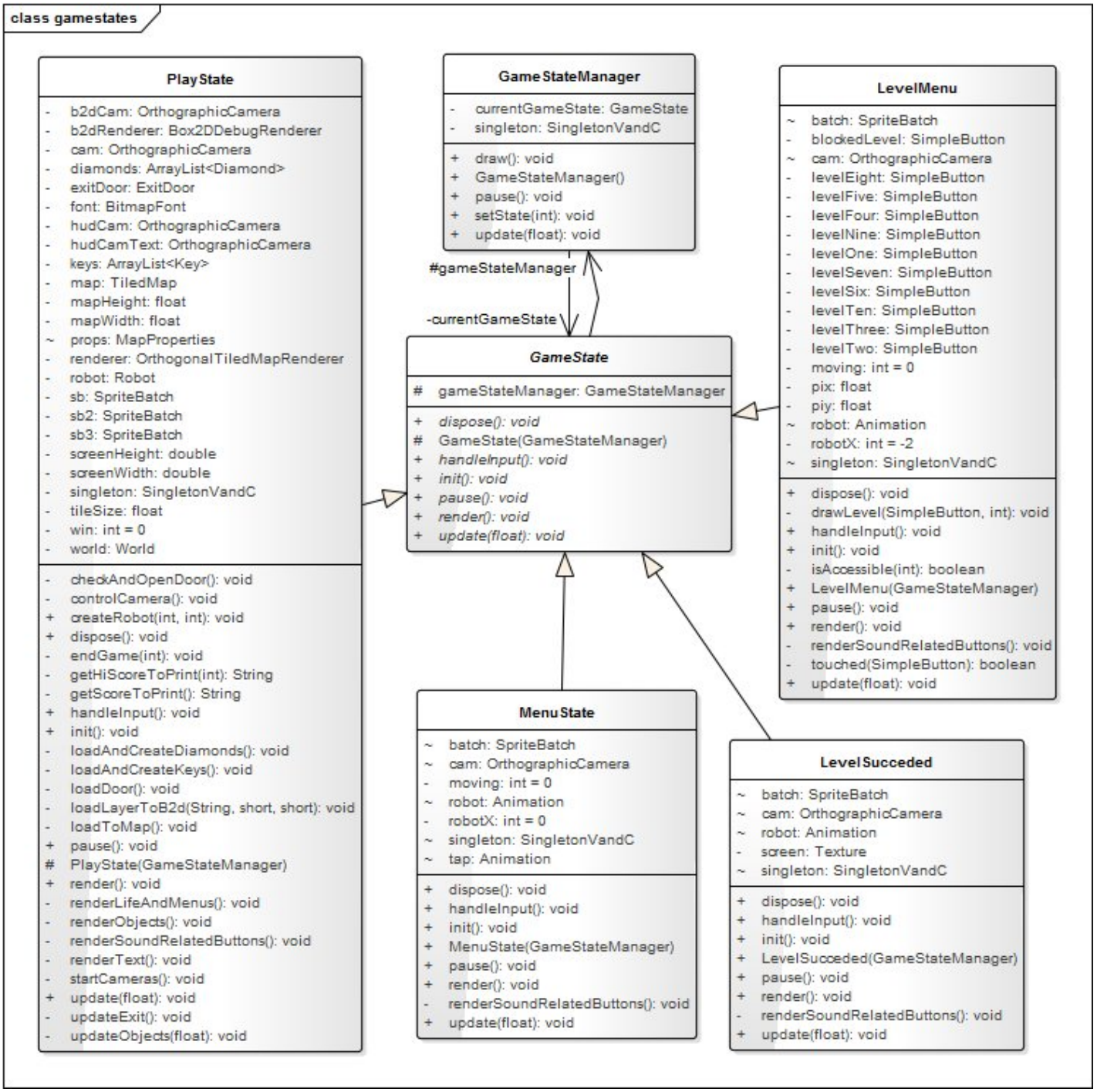
3.1 Estrutura de packages

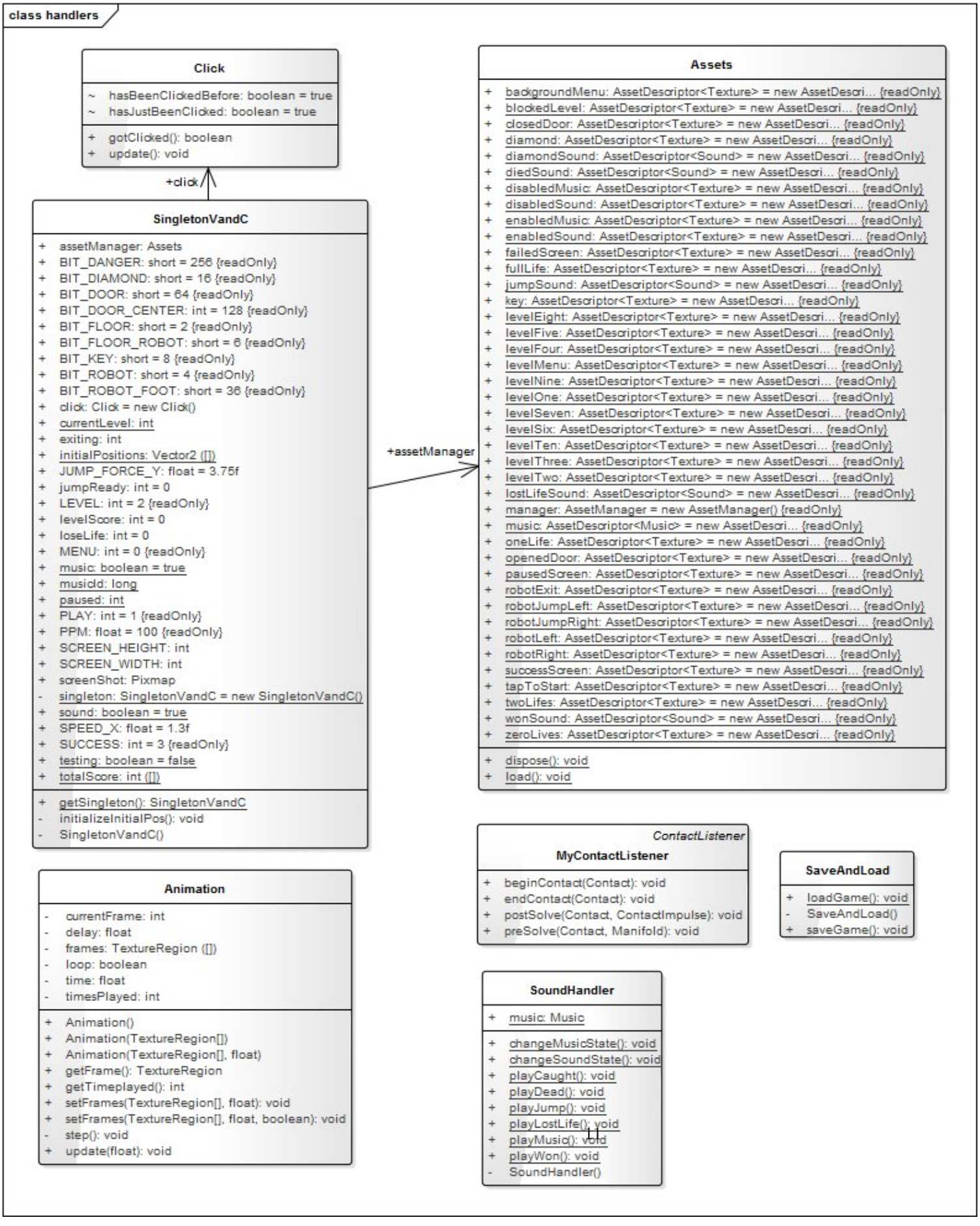


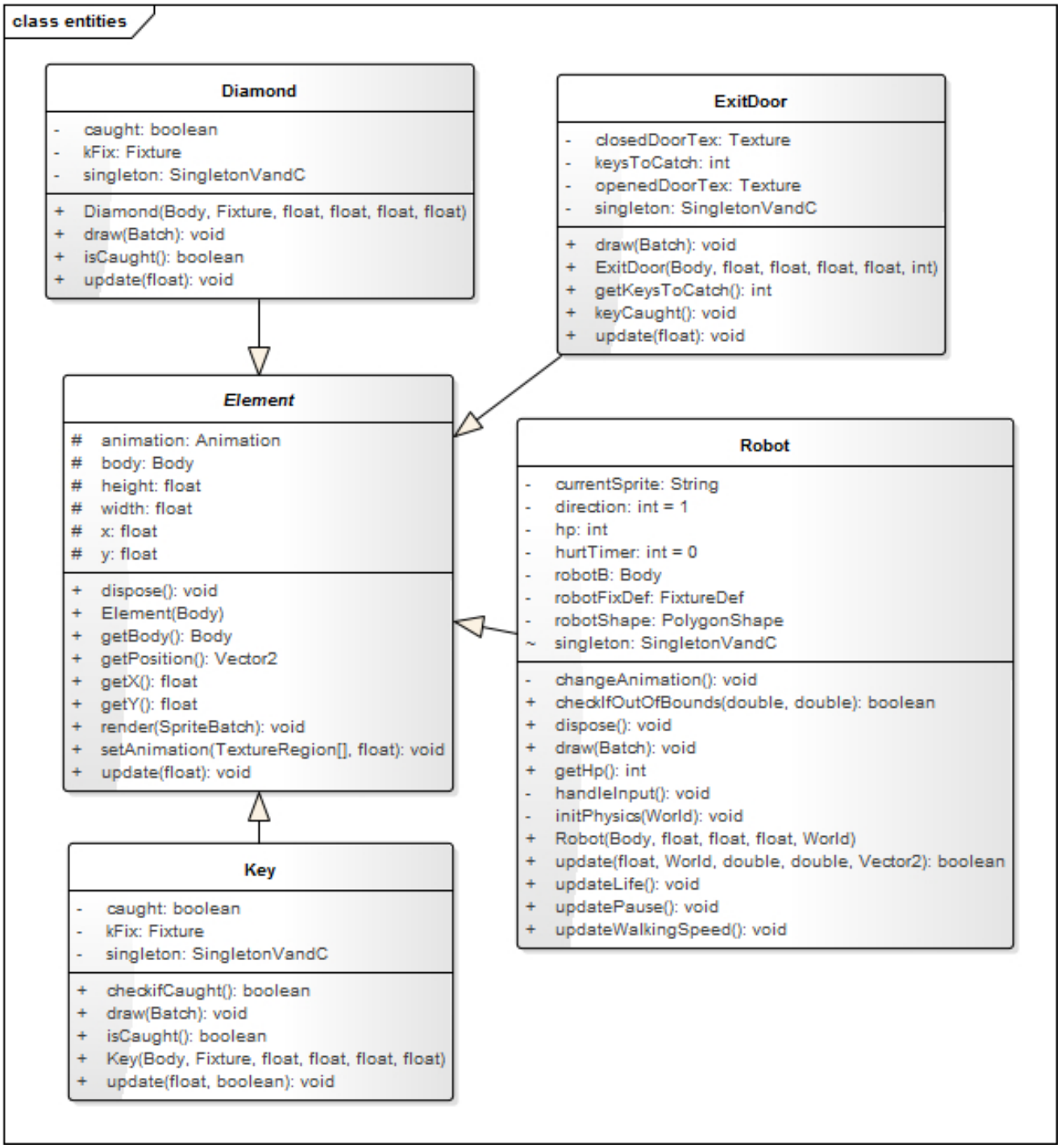
3.2 Estrutura de classes






class gamestates
PlayState
- b2dCam: OrthographicCamera
- b2dRenderer: Box2DDebugRenderer
- cam: OrthographicCamera
- diamonds: ArrayList<Diamond>
- exitDoor: ExitDoor
- font: BitmapFont
- hudCam: OrthographicCamera
- hudCamText: OrthographicCamera
- keys: ArrayList<Key>
- map: TiledMap
- mapHeight: float
- mapWidth: float
~ props: MapProperties
- renderer: OrthogonalTiledMapRenderer
- robot: Robot
- sb: SpriteBatch
- sb2: SpriteBatch
- sb3: SpriteBatch
- screenHeight: double
- screenWidth: double
- singleton: SingletonVandC
- tileSize: float
- win: int = 0
- world: World
- checkAndOpenDoor(): void
- controlCamera(): void
+ createRobot(int, int): void
+ dispose(): void
- endGame(int): void
- getHiScoreToPrint(int): String
- getScoreToPrint(): String
+ handleInput(): void
+ init(): void
- loadAndCreateDiamonds(): void
- loadAndCreateKeys(): void
- loadDoor(): void
- loadLayerToB2d(String, short, short): void
- loadToMap(): void
+ pause(): void
# PlayState(GameStateManager)
+ render(): void
- renderLifeAndMenus(): void
- renderObjects(): void
- renderSoundRelatedButtons(): void
- renderText(): void
- startCameras(): void
+ update(float): void
- updateExit(): void
- updateObjects(float): void
GameStateManager
- currentGameState: GameState
- singleton: SingletonVandC
+ draw(): void
+ GameStateManager()
+ pause(): void
+ setState(int): void
+ update(float): void
#gameStateManager
-currentGameState
GameState
gameStateManager: GameStateManager
+ dispose(): void
# GameState(GameStateManager)
+ handleInput(): void
+ init(): void
+ pause(): void
+ render(): void
+ update(float): void
LevelMenu
~ batch: SpriteBatch
- blockedLevel: SimpleButton
~ cam: OrthographicCamera
- levelEight: SimpleButton
- levelFive: SimpleButton
- levelFour: SimpleButton
- levelNine: SimpleButton
- levelOne: SimpleButton
- levelSeven: SimpleButton
- levelSix: SimpleButton
- levelTen: SimpleButton
- levelThree: SimpleButton
- levelTwo: SimpleButton
- moving: int = 0
- pix: float
- piy: float
~ robot: Animation
- robotX: int = -2
~ singleton: SingletonVandC
+ dispose(): void
- drawLevel(SimpleButton, int): void
+ handleInput(): void
+ init(): void
- isAccessible(int): boolean
+ LevelMenu(GameStateManager)
+ pause(): void
+ render(): void
- renderSoundRelatedButtons(): void
- touched(SimpleButton): boolean
+ update(float): void
MenuState
~ batch: SpriteBatch
~ cam: OrthographicCamera
- moving: int = 0
~ robot: Animation
- robotX: int = 0
~ singleton: SingletonVandC
~ tap: Animation
+ dispose(): void
+ handleInput(): void
+ init(): void
+ MenuState(GameStateManager)
+ pause(): void
+ render(): void
- renderSoundRelatedButtons(): void
+ update(float): void
LevelSucceded
~ batch: SpriteBatch
~ cam: OrthographicCamera
~ robot: Animation
- screen: Texture
~ singleton: SingletonVandC
+ dispose(): void
+ handleInput(): void
+ init(): void
+ LevelSucceded(GameStateManager)
+ pause(): void
+ render(): void
- renderSoundRelatedButtons(): void
+ update(float): void


class handlers
Click
~ hasBeenClickedBefore: boolean = true
~ hasJustBeenClicked: boolean = true
+ gotClicked(): boolean
+ update(): void
+click
SingletonVandC
+ assetManager: Assets
+ BIT_DANGER: short = 256 {readOnly}
+ BIT_DIAMOND: short = 16 {readOnly}
+ BIT_DOOR: short = 64 {readOnly}
+ BIT_DOOR_CENTER: int = 128 {readOnly}
+ BIT_FLOOR: short = 2 {readOnly}
+ BIT_FLOOR_ROBOT: short = 6 {readOnly}
+ BIT_KEY: short = 8 {readOnly}
+ BIT_ROBOT: short = 4 {readOnly}
+ BIT_ROBOT_FOOT: short = 36 {readOnly}
+ click: Click = new Click()
+ currentLevel: int
+ exiting: int
+ initialPositions: Vector2 ([])
+ JUMP_FORCE_Y: float = 3.75f
+ jumpReady: int = 0
+ LEVEL: int = 2 {readOnly}
+ levelScore: int = 0
+ loseLife: int = 0
+ MENU: int = 0 {readOnly}
+ music: boolean = true
+ musicId: long
+ paused: int
+ PLAY: int = 1 {readOnly}
+ PPM: float = 100 {readOnly}
+ SCREEN_HEIGHT: int
+ SCREEN_WIDTH: int
+ screenShot: Pixmap
- singleton: SingletonVandC = new SingletonVandC()
+ sound: boolean = true
+ SPEED_X: float = 1.3f
+ SUCCESS: int = 3 {readOnly}
+ testing: boolean = false
+ totalScore: int ([])
+ getSingleton(): SingletonVandC
- initializeInitialPos(): void
- SingletonVandC()
+assetManager
Assets
+ backgroundMenu: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ blockedLevel: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ closedDoor: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ diamond: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ diamondSound: AssetDescriptor<Sound> = new AssetDescri... {readOnly}
+ diedSound: AssetDescriptor<Sound> = new AssetDescri... {readOnly}
+ disabledMusic: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ disabledSound: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ enabledMusic: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ enabledSound: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ failedScreen: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ fullLife: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ jumpSound: AssetDescriptor<Sound> = new AssetDescri... {readOnly}
+ key: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelEight: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelFive: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelFour: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelMenu: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelNine: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelOne: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelSeven: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelSix: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelTen: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelThree: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ levelTwo: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ lostLifeSound: AssetDescriptor<Sound> = new AssetDescri... {readOnly}
+ manager: AssetManager = new AssetManager() {readOnly}
+ music: AssetDescriptor<Music> = new AssetDescri... {readOnly}
+ oneLife: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ openedDoor: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ pausedScreen: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ robotExit: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ robotJumpLeft: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ robotJumpRight: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ robotLeft: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ robotRight: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ sucessScreen: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ tapToStart: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ twoLifes: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ wonSound: AssetDescriptor<Sound> = new AssetDescri... {readOnly}
+ zeroLives: AssetDescriptor<Texture> = new AssetDescri... {readOnly}
+ dispose(): void
+ load(): void
Animation
- currentFrame: int
- delay: float
- frames: TextureRegion ([])
- loop: boolean
- time: float
- timesPlayed: int
+ Animation()
+ Animation(TextureRegion[])
+ Animation(TextureRegion[], float)
+ getFrame(): TextureRegion
+ getTimeplayed(): int
+ setFrames(TextureRegion[], float): void
+ setFrames(TextureRegion[], float, boolean): void
- step(): void
+ update(float): void
ContactListener
MyContactListener
+ beginContact(Contact): void
+ endContact(Contact): void
+ postSolve(Contact, ContactImpulse): void
+ preSolve(Contact, Manifold): void
SaveAndLoad
+ loadGame(): void
- SaveAndLoad()
+ saveGame(): void
SoundHandler
+ music: Music
+ changeMusicState(): void
+ changeSoundState(): void
+ playCaught(): void
+ playDead(): void
+ playJump(): void
+ playLostLife(): void
+ playMusic(): void
+ playWon(): void
- SoundHandler()

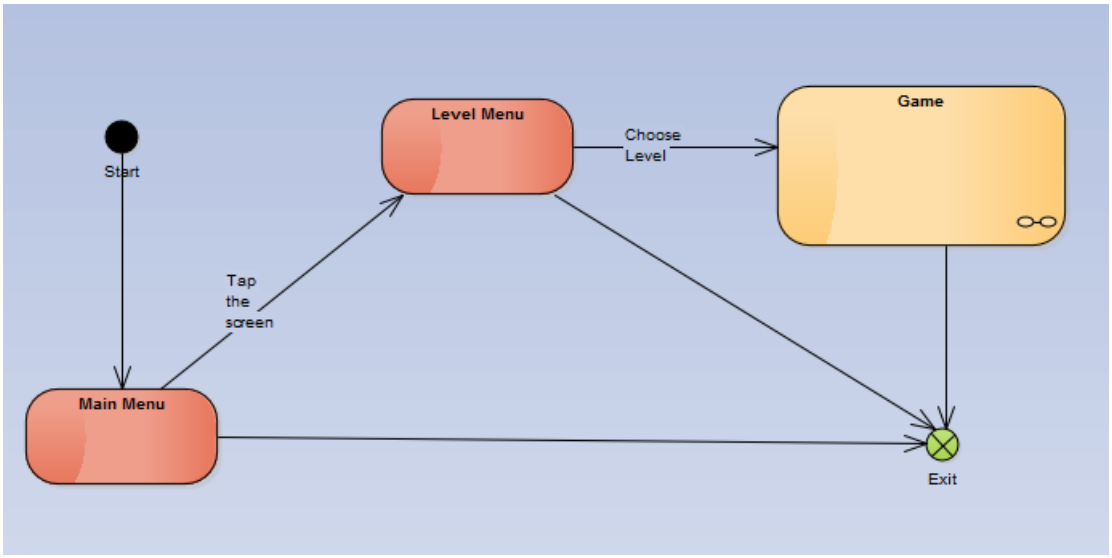

class entities
Diamond
- caught: boolean
- kFix: Fixture
- singleton: SingletonVandC
+ Diamond(Body, Fixture, float, float, float, float)
+ draw(Batch): void
+ isCaught(): boolean
+ update(float): void
ExitDoor
- closedDoorTex: Texture
- keysToCatch: int
- openedDoorTex: Texture
- singleton: SingletonVandC
+ draw(Batch): void
+ ExitDoor(Body, float, float, float, float, int)
+ getKeysToCatch(): int
+ keyCaught(): void
+ update(float): void
Element
animation: Animation
body: Body
height: float
width: float
x: float
y: float
+ dispose(): void
+ Element(Body)
+ getBody(): Body
+ getPosition(): Vector2
+ getX(): float
+ getY(): float
+ render(SpriteBatch): void
+ setAnimation(TextureRegion[], float): void
+ update(float): void
Robot
- currentSprite: String
- direction: int = 1
- hp: int
- hurtTimer: int = 0
- robotB: Body
- robotFixDef: FixtureDef
- robotShape: PolygonShape
~ singleton: SingletonVandC
- changeAnimation(): void
+ checkIfOutOfBounds(double, double): boolean
+ dispose(): void
+ draw(Batch): void
+ getHp(): int
- handleInput(): void
- initPhysics(World): void
+ Robot(Body, float, float, float, World)
+ update(float, World, double, double, Vector2): boolean
+ updateLife(): void
+ updatePause(): void
+ updateWalkingSpeed(): void
Key
- caught: boolean
- kFix: Fixture
- singleton: SingletonVandC
+ checkifCaught(): boolean
+ draw(Batch): void
+ isCaught(): boolean
+ Key(Body, Fixture, float, float, float, float)
+ update(float, boolean): void

3.3 Padrões de desenho utilizados

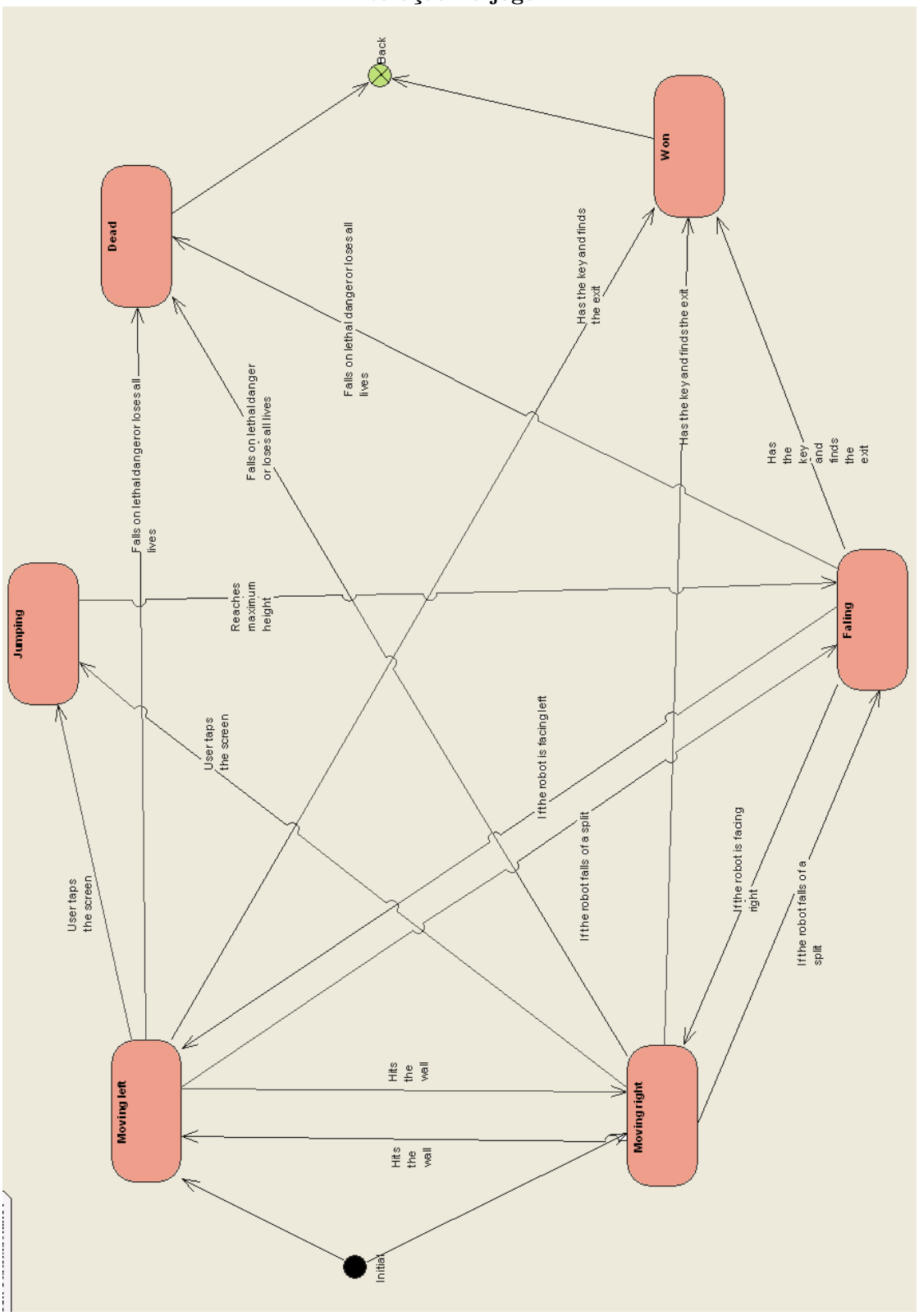
- *Creational Design Patters*
  - **Singleton** : *Usado na Classe SingletonVandC, onde estão reunidas todas as variáveis estáticas necessárias à aplicação.*
  - **Object Pool** : *Usado, por exemplo, na Classe Assets. Para evitar criar várias instâncias de um AssetManager, é apenas criado um ao qual podem aceder todas as classes. Podem também aceder às funções da Classe Assets.*
- *Structural design patterns*
  - **Private Class** : *Quase todas as classes utilizam este design pattern.*
- *Behavioral design patterns*
  - **Mediator** : *É utilizado, por exemplo, quando um diamante é apanhado pelo jogador para que o boleano que controla o estado de cada diamante se altere.*
  - **State** : *Utilizado para controlar todos os estados de jogo no package jumpyjay.gamestates*

3.4 Mecanismos

Interação entre menus



Interação no jogo



3.5 Bibliotecas e ferramentas utilizadas

- **LibGDX** - *http://libgdx.badlogicgames.com/*
- **Box2D** - *http://box2d.org/*
- **Tiled Maps** - *http://www.mapeditor.org/*
- **Gradle** - *https://gradle.org/*
- **Enterprise Architect** - *http://www.sparxsystems.com.au/*
- **ObjectAid** - *http://www.objectaid.com/*

3.6 Testes Manuais

Para alem dos testes em JUnit testamos a aplicação manualmente, fazendo o seguinte:

- Ir contra um dos lados do mapa e verificar que a camara não sai.
- Dar um toque no ecrã enquanto o robot está no ar para verificar que não salta de novo.
- Verificar que o jogo responde sempre ao toque, seja em qualquer sitio do ecrã, ou numa parte do ecrã em especifico (*botões mute/unmute, escolher um nivel, etc).*
- Tentar escolher um nível bloqueado e verificar que não dá.
- Verificar que o *highscore* é sempre atualizado quando ultrapassado.
- Sair do jogo e voltar a entrar para saber se o progresso e os respectivos *highscores* de cada nível são guardados.

3.7 Dificuldades encontradas

Ao longo do desenvolvimento deste jogo fomos encontrando algumas dificuldades, que acabaram por ser sempre ultrapassadas. Primeiramente tivemos um **problema com as animações**, sendo que não conseguiamos mostrar corretamente o mapa no ecrã quando eram adicionadas animações ao *robot*.

Quando começamos a testar o programa em *Android* deparamo-nos com um **erro ao sair do jogo** que era apresentado na consola (*AL lib: alc cleanup: 1 device not closed*) e causava que o jogo deixasse de responder no telemovel.

Após a conclusão do jogo em si, existiu ainda outra dificuldade que acabamos por conseguir ultrapassar que foi na **execução dos testes unitários** pois não saberiamos como criar testes unitários ao usar a Box2D e o LibGDX.

3.8 Resolução

Em primeiro lugar, o problema com as animações deveu-se ao facto de estarmos a utilizar uma classe *Animation* existente. Tendo em conta os problemas enfrentados e sabendo aquilo que queríamos que a classe fizesse e tivesse, optámos por adaptar uma classe *Animation* para que esta viesse de encontro àquilo que procurávamos, resolvendo assim o problema encontrado.

Em segundo lugar, deparámo-nos com um erro de acesso a memória. Exigiu algum tempo para detetar a sua origem mas ao fim de algum tempo chegámos à conclusão que se devia ao facto de estarmos a fazer *"dispose"* de um objeto a que íamos aceder seguidamente. Para a resolução deste problema foi-nos muito útil a prespectiva de *Debug* do IDE Eclipse;

Por fim, encontrámos uma grande dificuldade em fazer testes unitários incorporando o motor de física Box2D e o LibGDX. Acabámos por conseguir alterar as dependências por forma a realizar testes que incorporassem o motor de física, o que nos permitiu testar algumas funcionalidades dos nossos elementos de jogo.

4 Conclusão

4.1 Cumprimento dos objetivos

Inicialmente tínhamos como objetivo incluir no jogo a oportunidade de jogar em *multiplayer*, no entanto, ao longo do desenvolvimento do trabalho chegamos à conclusão de que este jogo em particular não seria um bom jogo para ser jogado por múltiplas pessoas competindo entre elas ao mesmo tempo, então decidimos deixar essa ideia de parte e focarmo-nos em partes mais fulcrais do jogo como as animações ou mesmo a física de jogo. No entanto, podemos dizer que os restantes objetivos para esta aplicação foram concluídos com sucesso.

4.2 Melhorias possíves

Uma melhoria possível que se poderia implementar no nosso jogo seria a integração com o sistema de pontuações da *Play Store* permitindo assim, ao jogador, guardar o seu highscore na sua conta *Google* e partilha-la com o resto dos utilizadores.

Também queremos disponibilizar mais níveis e acrescentar uma historia ao jogo.

4.3 Contribuição dos elementos do grupo

Este trabalho foi sendo desenvolvido sempre em conjunto, sendo que todos os elementos participaram na execução do mesmo.

4.4 Notas finais

Concluindo, queremos agradecer ao criador do jogo *Jack N' Jill*, Rohan Narang, que para além de nos ter dado uma inspiração para a criação do *Jumpy Jay* também se mostrou disponível para nos explicar e ajudar no desenvolvimento do jogo, dando-nos dicas e acima de tudo criando as *sprites* especialmente para o *Jumpy Jay*, que são uma parte fundamental deste jogo.

5 Referências

1. Mora, William. "LibGDX Tutorial - A Running Game With LibGDX" *William Mora Blog*. 07 Junho 2014. Web. - *http://williammora.com/a-running-game-with-libgdx-part-1/*

2. Bruner, Nick. "Introduction to Tiled Map Editor: A Great, Platform-Agnostic Tool for Making Level Maps - Tuts+ Game Development Tutorial" *Game Development Tuts+*. 10 Dezembro 2012. Web. *http://gamedevelopment.tutsplus.com/tutorials/introduction-to-tiled-map-editor-a-great-platform-agnostic-tool-for-making-level-maps–gamedev-2838*

3. "LibGDX Tutorial series" *Games From Scratch*. Setembro 2013. Web. - *http://www.gamefromscratch.com/page/LibGDX-Tutorial-series.aspx*

4. Srivastava, Rahul. "Using Box2D in LibGDX Game" *Rotating Canvas Games*. 14 Abril 2012. Web. - *http://rotatingcanvas.com/using-box2d-in-libgdx-game-part-i/*

5. Javaid, Shahmir. "JUnit with LibGDX Using Gradle." *Shahmirj Blog*. 11 Julho 2014. Web. - *http://shahmirj.com/blog/getting-junit-working-with-libgdx-in-gradle*