2018

Relatório Trabalho Pratico

Computação Móvel

Fernando Camilo Nº13233

Diogo Portela Nº13236

Bruno Couto Nº10664

1/15/2018

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Introdução

No âmbito da unidade curricular de computação móvel, foi proposto ao à turma de 2º ano de EDJD, que elabora-se um trabalho prático usando a biblioteca gráfica “libGDX”. Sendo que esse trabalho seria a idealização e implementação de um videojogo para a plataforma "android". Não havendo restrições quanto ao jogo que poderia ser feito, e após uma longa deliberação, o grupo de trabalho decidiu fazer um jogo de gestão de restaurantes. Em que o objectivo seria construir um restaurante, desde o edifício até ao “staff” que lá trabalharia. Nesta build que entregamos juntamente com este relatório, apenas é referente a uma fase inicial do jogo, isto porque as restrições temporais e a divisão de atenção pelas outra UC deste semestre não permitiram avançar mais na elaboração deste, mas este projecto será continuado no futuro.

Dito isto à que referir agora a estruturadeste relatório, que ira descrever ao pormenor tudo o que foi feito até ao momento, desde os recursos usados até aos algoritmos implementados, e servirá como guia do projecto. Este trabalho como já foi referido foi elaborado para a plataforma “*android*” e a linguagem de programação usada foi “Java”, usando a “API” “Android Studio”.

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
Licenciatura em Engenharia de desenvolvimento de jogos de digitais
2017/2018

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

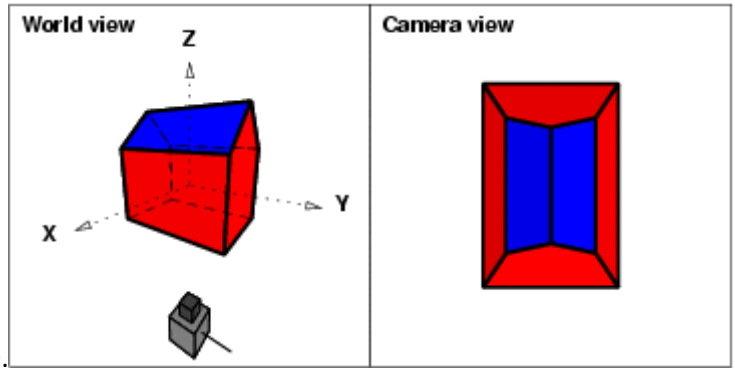
Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

LibGDX

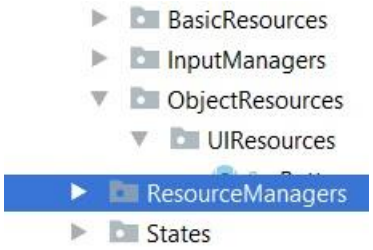
Esta "framework" gráfica foi lançado com este nome, em 2013 por Mario Zechner, que era um programador que tinha intenções de desenvolver jogos para "android", mas verificou que era difícil testar no computador a aplicação então quis ligar as duas plataformas.

Esta biblioteca tem como "backup" uma versão optimizada de "OpenGL", para nunca ser necessário ter liderar directamente com esta "API", assim sendo esta "framework" foi idealizada em 3D, apesar de imolar 2D por sobreposição de objectos e uma câmara com uma vista superior que dá a ilusão de 2D.



Organização do projeto

O projecto esta quase na sua totalidade organizado em pastas, começando pela pasta "BasicResources", que alberga classes maioritariamente de suporte, os alicerces, do projecto, depois temos a pasta que guarda a gestão de "Input" de jogo, chamada de "InputManagers", as duas ultimas são a "ObjectResources", que tem o "folder" "UIResources" que guarda os elementos usados no menu de jogo, em que nestas pastas são guardadas as classes que modelam todos os objectos que aparecem no ecrã de jogo e por ultimo a pasta "States" que guarda todos os estados de jogo possíveis dentro do jogo.



ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe Date

Nesta classe apenas se constrói um relógio de jogo para ponto de referência do jogador, com tempo e data para fazer face a incompatibilidade do libGdx com o que já existe em "Java", para os objectos de jogo.

Classe GameViewPort

Classe criada com o objectivo de manter uniforme o tamanho do mapa de jogo independentemente do tamanho do ecrã do telemóvel em que esta a correr o jogo. Isto é conseguido ao multiplicar o tamanho do mapa pelo tamanho do ecrã usado.

```
public GameViewport(Point screenPosition, Vector2 screenSize, float worldWidth, float worldHeight, Camera
    float widthRatio = screenSize.x / (float)Gdx.graphics.getBackBufferWidth();
    float heightRatio = screenSize.y / (float)Gdx.graphics.getBackBufferHeight();
    worldWidthPermanent = worldWidth;
    worldHeightPermanent = worldHeight;

    setWorldSize( worldWidth: worldWidth * widthRatio,  worldHeight: worldHeight * heightRatio);
    setScreenBounds(screenPosition.X, screenPosition.Y, (int)screenSize.x, (int)screenSize.y);
    setCamera(camera);
}
```

Classe GraphGrid

Classe que representa uma grelha de "nodes" em que podemos procurar caminhos (grafo), tendo uma grelha marca quais os "tiles" podemos caminhar ou não. Os vários construtores são para gerar o grafo de acordo com o mapa introduzido ou gerado e a função "GetNeighbors" serve para retornar os vizinhos de um "node" particular.

```
public List<Node> GetNeighbors(Node node)
{
    List<Node> neighbors = new ArrayList<>();

    for (int x = -1; x <= 1; x++){
        for (int y = -1; y <= 1; y++){
            if (x == 0 && y == 0)
                continue;

            int checkX = node.gridX + x;
            int checkY = node.gridY + y;

            if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY)
            {
                neighbors.add(nodes[checkX][checkY]);
            }
        }
    }

    return neighbors;
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe Grid

Classe implementada para suportar o mapa de jogo, ou seja esta classe gera o mapa de jogo, com uma escala fixa e o tipo de célula que pertence a cada coordena de mapa.

Uma das funções é usada para calcular os cantos da grelha, outra que verifica as células que fazem "overlaping", sendo que esta classe para alem de gerar o mapa também é responsável por permitir a construção em tempo real do restaurante, apesar de esta componente não estar ainda completa, isto é conseguido ao verificar limites e se existem células do mesmo tipo em cima uma da outra e caso haja substituí por outra que da ilusão de formação de paredes interligadas.

```
    }
    for (GridCell[] row : grid.cells) {
        for (GridCell cell : row) {
            if (cell != null) {
                int x = (int) cell.getPosition().x - newMinX;
                int y = (int) cell.getPosition().y - newMinY;
                newArray[x][y] = cell;
            }
        }
    }
    this.cells = newArray;
    for (GridCell cell : grid.wallList) {
        if (!wallList.contains(cell))
            wallList.add(cell);
    }
    for (GridCell cell : grid.cornerList) {
        if (!cornerList.contains(cell))
            cornerList.add(cell);
    }
    calculateExternalWallsOnArea(minIntersect, maxIntersect);
```

Classe Map

É apenas a classe que guarda o mapa de jogo em grelha.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

```
    }
    for (GridCell[] row : grid.cells) {
        for (GridCell cell : row) {
            if (cell != null) {
                int x = (int) cell.getPosition().x - newMinX;
                int y = (int) cell.getPosition().y - newMinY;
                newArray[x][y] = cell;
            }
        }
    }
    this.cells = newArray;
    for (GridCell cell : grid.wallList) {
        if (!wallList.contains(cell))
            wallList.add(cell);
    }
    for (GridCell cell : grid.cornerList) {
        if (!cornerList.contains(cell))
            cornerList.add(cell);
    }
    calculateExternalWallsOnArea(minIntersect, maxIntersect);
```

Classe Node

Classe que representa um nodo no grafo que contém o preço, o custo, custo da heurística e o "*parent node*". Tem dois construtores, um para gerar um nodo com preço para caminhar e outro para criar o mesmo nodo através de um “boolean”(“walkable” ou “nonwalkable”). Por último tem também uma função para retornar o custo final do nodo.

Classe Pathfinding

Classe com o algoritmo A* de forma a encontrar o melhor caminho entre ponto A ao ponto B num "tile map". O algoritmo, pode depender do pedido e ou da geração do mapa, e o custo para passar de um nodo para outro, ou até pode ter valores predefinidos.

```
openSet.add(startNode);

while (openSet.size() > 0) {
    Node currentNode = openSet.get(0);
    for (int i = 1; i < openSet.size(); i++) {
        if (openSet.get(i).fCost() < currentNode.fCost() || openSet.get(i).fCost() == currentNode.fCost() && openSet.get(i).hCost < currentNode.hCost) {
            currentNode = openSet.get(i);
        }
    }
    openSet.remove(currentNode);
    closedSet.add(currentNode);

    if (currentNode == targetNode) {
        return RetracePath(graphGrid, startNode, targetNode);
    }

    for (Node neighbor : graphGrid.GetNeighbors(currentNode)) {
        if (!neighbor.walkable || closedSet.contains(neighbor)) {
            continue;
        }

        int newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbor) * (ignorePrices ? 1 : (int) (10.f * neighbor.price));
        if (newMovementCostToNeighbour < neighbor.gCost || !openSet.contains(neighbor)) {
            neighbor.gCost = newMovementCostToNeighbour;
            neighbor.hCost = GetDistance(neighbor, targetNode);
            neighbor.parent = currentNode;

            if (!openSet.contains(neighbor))
            {
                openSet.add(neighbor);
            }
```

Classe Point

Classe apenas com o objectivo de permitir um ponto no espaço com duas dimensões do tipo inteiro.

Classe InputManager

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

É classe base para todo o "input" que surgira no decorrer do jogo, é do tipo abstracto e todas as outras classes com o objectivo de recolher a informação de toque no ecrã derivam desta.
Guarda valores base para fazer "zoom" ou também para a velocidade de movimento da câmara.

Classe GameInput

Nesta classe é implementado o código que trata dos vários tipos de acções que se pode ter com o ecrã, do dispositivo móvel, como "zoom" que é efectuado ao calcular a diferença entre a distância inicial e final entre os dedos do jogador aquando do gesto, sendo que este valor é controlado pela velocidade previamente definida na classe mãe desta e limitada por valores de "bounds". Quanto o gesto de "pan" é definido pelo gesto de arrasto dos dedos pelo ecrã e esse valor de translação é guardado, sendo que lhe é aplicado o valor de velocidade base. Por ultimo é modelado o movimento de toque do ecrã, "tap" que apenas verifica a posição onde foi aplicada a pressão e verifica o que foi pressionado.

```
public boolean tap(float x, float y, int count, int button) {

    if (x > GameState.getCurrentMenuSizeScreen()) {            //SE ESTIVER NO VIWEPORT.
        Vector2 touchedPositionOnWorld = new Vector2(GameState.currentViewport.unproject(new Vector2(x, y)));
        TouchableObject touchableObjectSelectedThisFrame = currentState.findTouchedObject(touchedPositionOnWorld);
        if (currentState.getSelectedObject() == null)    //SE NAO HOUVER NADA SELECIONADO.
        {
            currentState.setSelectedObject(touchableObjectSelectedThisFrame);
            if (touchableObjectSelectedThisFrame == null) {    //SE NAO HOUVE SELECÇAO AGORA.
                //select ground floor.
            }
        } else {                                        //SE HOUVER ALGO SELECIONADO
            //SE CARREGARES NO CHAO MOVE.
            if (touchableObjectSelectedThisFrame == null) {
                ((GameObject) currentState.getSelectedObject()).act(touchedPositionOnWorld);
            }
            //SE CARREGARES NOUTRO BONECO SELECIONA-O.
            else if (touchableObjectSelectedThisFrame instanceof Client || touchableObjectSelectedThisFrame instanceof Employee) {
                currentState.setSelectedObject(touchableObjectSelectedThisFrame);
            } else if(touchableObjectSelectedThisFrame instanceof Item){
                if(currentState.getSelectedObject() instanceof Client){
                    if(((Table)touchableObjectSelectedThisFrame).isUsed() == false){
                        ((Table)touchableObjectSelectedThisFrame).setUsed(true);
                        ((Client)currentState.getSelectedObject()).act(touchableObjectSelectedThisFrame);
                    }
                }
            }
        }
    } else {                                            //SE ESTIVER NO MENU.
        Vector2 touchedPositionOnWorld = new Vector2(GameState.currentMenuViewport.unproject(new Vector2(x, y)));
        TouchableObject touchableObjectSelectedThisFrame = currentState.findTouchedObject(touchedPositionOnWorld);
        if (touchableObjectSelectedThisFrame instanceof Button)
            ((Button) touchableObjectSelectedThisFrame).onClick();
```

Typo: In word 'SELECIONADO' more... (Ctrl+F1)

Classe MenuInput

Implementa os mesmos métodos da classe "GameInput" mas verifica se é um botão do menu e caso toma a acção de que um botão foi pressionado.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe BuildInput

Implementa os métodos anteriormente descritos, mas com o objectivo ligeiramente diferente, o de toque guarda a posição e célula que foi pressionada para usar essa posição no método “panStop” e nessa distância diagonal entre os movimentos é definida uma nova grelha que é passa a ser construída nesse espaço.

```
@Override
public boolean panStop(float x, float y, int pointer, int button) {
    if (isBuildingWalls) {
            Vector2 touchedPositionOnWorld = new Vector2(GameState.currentViewport.unproject(new Vector2(x, y)));
        Grid newGrid = new Grid(Grid.GridType.interior, firstPosition, touchedPositionOnWorld);
        return true;
    } else {
```

Classe Room

Classe base para os espaços que seriam criados e personalizados pelo jogador de forma a definir o fluxo de cada quarto e no final o fluxo de serviço do restaurante. Esta classe e todas as que de si variam, apenas guardavam os valores que iriam definir cada espaço do restaurante. Valores como limites máximos de cada quarto de acordo com um valor máximo de dinheiro que se podia gastar em cada, tal como os limites físicos que estes podiam ter dentro do restaurante, sendo que seria este mesmo tamanho que iria definir a capacidade de trabalho de cada “room”.
Outra variante desta é a classe “RoomAsActor” que tem o mesmo intuito da classe anterior mas tentado usar uma classe originária do “libGDX”. As classe “Kitchen”, “MainHall” e “Bathroom” variam da “room” guardando os objectos que iria conter dentro.
Por ultimo a classe “restaurant” iria albergar todos os “rooms” de forma a formar um restaurante e a “restaurantBuilding” tinha o objectivo de definir os limites de construção do restaurante tanto no interior como exterior.

Classe InventoryChoices

Classes que iriam ser usadas na fase de compra de inventário antes do serviço do restaurante, apenas contem dicionários que guardam os ingredientes que iriam ser usados e tal como as suas quantidades e valores a pagarem pelo jogador.

Classe TouchableObject

Classe abstracta de onde vão derivar as outras que modelem objectos capazes de serem seleccionados.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe GameObject

Estende da classe referida anteriormente de forma também a funcionar como base para qualquer objecto que a herde. Assim sendo qualquer objecto deste tipo pode ser seleccionado, tem uma posição, “sprite” e escala que o define, aqui é feito o “render” da textura que vem da “sprite”, tal como a destruição dessa textura da memória do dispositivo para tornar o jogo optimizado. A classe “item” tem o mesmo objectivo mas para objectos sem movimento autónomo como fogões ou armários. O único exemplo desta classe é na “Table” que é usada como objecto de destino dos clientes.

Classe Employee

Classe simples de extensão da “gameObject” em que a grande diferença é que um objecto desta classe tem a capacidade de movimento, iria ser usada na construção do “staff” que iria estar no restaurante.

Classe GridCell

Usada para guardar os tipos de células presentes no jogo de forma a ser aplicada a textura correspondente.

Classe Finance

Nesta classe são guardadas todas as informações que ao sistema financeiro do jogo dizem respeito, desde de limites para o gasto do dinheiro inicial na fase de construção, ao retorno que seria expectável de se ter de acordo com o número de reservas que seriam simuladas aquando da fase de compra de inventário. Foi usada para calcular o dinheiro actual do jogador, verificando para todos os clientes presentes os estado deles e caso eles tenham sido atendidos a tempo então o dinheiro, que eles tinham seria somado ao dinheiro do jogador, mais o valor restante da capacidade de espera presente na classe “Client” que será referida mais adiante deste relatório. Este valor seria para simular o valor de uma “tip” grojeta que podia ser dado ao jogador pelo atendimento rápido, por ultimo o “score” do jogador também é aqui calculado e advém do valor do dinheiro actual que tem a

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

multiplicar pelo tempo de jogo que tem.

```
public void currentMoneyCalculation(float currentMoney, ArrayList<Client[]> clients) {
    for (Client[] c : clients
            ) {
        for (Client client : c
                ) {
            if (client.getPatience() > 0 && client.isLeaving())
                currentMoney += client.getMoney() + client.getPatience() / 10;
            else if (client.getPatience() <= 0 && client.isLeaving())
                currentMoney -= client.getMoney();
        }
    }
}

public void ScoreCalculation(Player player, float time) {
    float currentScore = player.getScore();
    if (!player.isGameOver()) {
        currentScore += (int) (player.income.currentMoney * time);
        player.setScore(currentScore);
```

Classe Client

A classe “Client” é das mais centrais de todo o projecto, visto que é sobre esta que se desenrola todo o conceito do jogo, deriva de “gameObject” portanto tem todas as suas características, tendo também um género (masculino, feminino), tem um nome, quatro estados possíveis, um valor de paciência que ira ser o valor de espera do cliente antes de ele passar do estado em que estava para um de “isLeaving” e assim caso tivesse o valor de paciência a zero o cliente ira embora e o dinheiro que tivesse seria descontado no valor do “income” do jogador. O nome é definido de maneira aleatória, de acordo com o seu género de forma a separar os clientes e formar ate casais e famílias, esse nome vem de uma lista de dez nomes para cada género e dez apelidos. Foi usada a ferramenta “Hiero” de forma a o nome acompanhar os clientes mas esta característica apesar de funcional não é esteticamente aceitável pois não foi possível arranjar uma “font” que permitisse atingir uma escala condizente com a das texturas do clientes.(no método render desta classe existe uma linha de código que se encontra comentada que permite que os nomes acompanhem os clientes). Por ultimo existe um método “update” que define o fluxo de estados do cliente, quando ele passa a estado de espera, a comer, e embora, tal como também é aplicado o algoritmo de A* para ele encontrar o caminho mais rápido possível para o destino. A forma visual que foi encontrada para demonstrar a perda de paciência para esperar é a de alteração da cor do cliente para vermelho até que atinja o estado em que se vá embora.

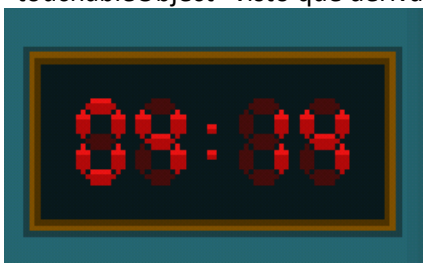
ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe Button

Deriva de classe “UIObject” e apenas modela um botão, tem o mesmo comportamento de um “touchableObject” visto que deriva deste por sequência da classe de “UI”



Classe Managers

As classes “fontManager” e “TextureManager” apenas guardam e gerem as "*fonts*" e texturas que estão presentes no jogo. Para as texturas guarda uma cópia da textura para ela ser usada ao longo de todo o jogo de forma a ter um uso optimizado enquanto o mesmo método tenha sido usado para as "*fonts*", o libGDX como é “open-source” mas que pode ser usado comercialmente, então não pode usar "*fonts*" predefinidas, pois estas são podem tornar o jogo vulnerável a direitos de “copy right “pelo texto usado no produto.

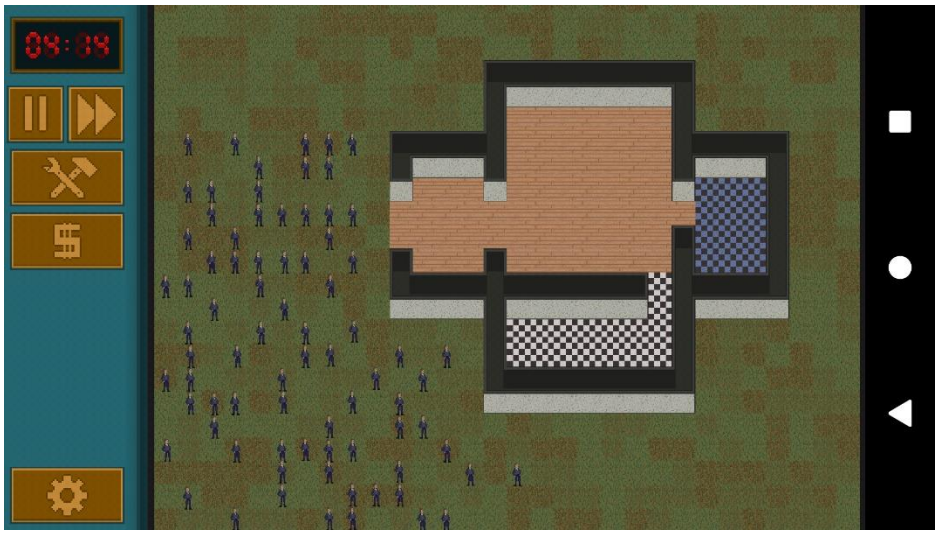
ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe ClientGenerator

Como os managers anteriormente apresentados mas para a classe "*Client*", que tem um dicionário com três "*keys*" para guardar nomes para os dois géneros e apelidos, sendo que estas estruturas são com o intuito de dar a ilusão de clientes que possam vir em família, casal, casados, etc. Sendo que os grupo são criados e colocados dentro de uma lista de "*arrays*" de clientes em que cada um representa um grupo destes. Os métodos presentes são os genéricos de "*draw*" e "*update*" para controlo dos clientes, quanto ao "*ClientCreator*" recebe os vários tipos de grupo de cliente e torna-os "*relatives*", varias funções que permitem dar um elemento aleatório a cada cliente desde o "*gender*" até à sua posição no mapa.



ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

```
public ClientGenerator() {
    random = new Random();
    numbRandom = random.nextInt( i: 9);
    clientsmaxNumb = 20;
    numbClients = clientsmaxNumb;
    nomes = new HashMap<Client.nameType, ArrayList<String>>();
    spriteBatch = new SpriteBatch();//Colocar as sprites dentro
    nomesMasculinos = new ArrayList<String>();
    PutNamesMasculino(nomesMasculinos);
    nomesFemininos = new ArrayList<String>();
    PutNamesFeminino(nomesFemininos);
    apelidos = new ArrayList<String>();
    PutNamesApelidos(apelidos);
    nomes.put(male, nomesMasculinos);
    nomes.put(female, nomesFemininos);
    nomes.put(apelido, apelidos);
    clients = new ArrayList<Client[]>(clientsmaxNumb);

    for (int i = 0; i < 100; i++) {
        clients.add((ClientCreator(single, nomes)));
    }
}

public void update(float deltaTime) {
    for (Client[] c : clients
            ) {
        for (Client client : c
                ) {
            client.update(deltaTime);
        }
    }
}

public void draw(SpriteBatch spriteBatch) {
    for (Client[] c : clients) {
        for (Client client : c) {
            client.render(spriteBatch);
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classes state

Classe abstracta "*state*" criada como base para trocar de ecrãs. Esta classe conte os objectos que podem ser "selecionaveis" em cada estado de jogo.
Contem também funções que podem ser rescritas de modo a acrescentar tipos de jogos seleccionáveis.
Desta classe derivam:
A classe abstracta "*GameState*" foi criada para todos os ecrãs durante o jogo. Esta classe serve principalmente para poder ter o ecrã dividido em dois, visto as outras não terem dois "*viewports*".
A classe usada para o decorrer do jogo em "*live mode*", para as personagens se moverem e o tempo correr.
"*BuildState*" é uma classe usada para editar o mapa e a "*Menustate*"classe usada para menu inicial, actualmente só tem um botão que leva ao jogo principal.

Classe player

Classe que serve para guardar todos os lados que mais tarde deviam serem recarregados ao fazer "load" do jogo. Esta classe inclui a câmara actual, o mapa, todos os "gameObjects" do jogo, a data, os dados de finanças, e o grafo para a IA.

```
//-------------------------Constructor-------------------------//
public Player(){
    gameCamera = new OrthographicCamera();
    currentMap = new Map( Width: 30,  Height: 30);
    allGameObjects = new ArrayList<GameObject>();
    date = new Date();
    graphPath = new GraphGrid();
    income= new Finance( initialMoney: 100,  inventoryValue: 0,  expectedIncome: 0);
    gameOver=false;
    score=0;
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Classe MainGame

Classe criada durante instanciação do LibGDX. Esta classe é a principal do jogo. É aqui que estão todos os dados considerados "estáticos", que sao transcendentes ao jogo actual, como o “player”, o “InputManager” e o “State”.

```
@Override
public void create() {
    TextureManager.Start();
    FontManager.Start();
    currentPlayer = new Player();
    testLabel = FontManager.loadFont( fontName: "ARIALUNI.TTF",  size: 10, Color.WHITE);
    batch = new SpriteBatch();

    setCurrentState(new MenuState());
}

@Override
public void render() {
    currentState.update(Gdx.graphics.getDeltaTime());
    Gdx.gl.glClearColor( red: 0.1f,  green: 0.1f,  blue: 0.1f,  alpha: 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    currentState.render(batch);
}

@Override
public void dispose() {
    batch.dispose();
    currentState.dispose();
    TextureManager.dispose();
}

@Override
public void resize(int width, int height) {
    super.resize(width, height);
    currentState.resize(width, height);
}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Computação Movel

Fernando Camilo Nº13233
Diogo Portela Nº13236
Bruno Couto Nº10664

Conclusão

A jeito de conclusão, fica à vista que apesar do trabalho ter sido longo e árduo a "build" apresentada esta longe de terminada, no Copeto geral conseguimos elaborar um "tapping game" em que o objectivo é servir o maior número de clientes sem eles irem embora para não atingir a banca rota.

Contudo a base do jogo de construção e gestão que idealizamos ao início está feita, apenas precisa de mais tempo para ser ajustada de forma a ter-mos em mãos um jogo de "build and manage".

Por falta de coordenação ou incompatibilidade de horários e ou dedicação não foi possível atingir o resultado final desejado mas o grupo espera que todo o trabalho que é visível em código seja tomado em conta na deliberação da nota final do trabalho pratico.

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
Licenciatura em Engenharia de desenvolvimento de jogos de digitais
2017/2018