



Engenharia Informática

Programação Avançada

Relatório

Trabalho Prático – Meta 1

Ano Letivo: 2018/2019

Curso: Engenharia Informática

Autores: Ricardo Pereira – 21250780; Tiago Recatia - 21250550

Disciplina: Programação Avançada

Turma: P3

Professores: José Marinho, Maria A. Correia

Data de Submissão: 28/04/2019

Índice

1. Decisões tomadas	3
1.1. Swap Crew Member	3
1.2. Lock In.....	3
1.3. Journey Tracker	3
1.4. Estados	3
2. Máquina de estados	4
3. Classes	5
3.1. StateAdapter.....	5
3.2. DataGame	5
3.3. Player	5
3.4. CrewMember	5
3.5. Ship.....	5
3.6. Room	5
3.7. Trap.....	5
3.8. Alien	6
3.9. DestinationEarth.....	6
3.10. TextUI.....	6
4. Relacionamento de classes.....	7
5. Funcionalidades e regras.....	8

1. Decisões tomadas

1.1. Swap Crew Member

Para o jogador escolher qual o membro da *crew* a realizar as ações, ou qual o *alien* a colocar num quarto foi desenvolvido um sistema de troca para alterar a personagem ou alien selecionado. Ou seja, se o crew member ativo for o primeiro e se o jogador escolher a opção **“Swap selected member”** o segundo membro fica ativado. No caso dos *aliens* se existirem mais de dois aliens para selecionar, a troca é feita com o alien seguinte na lista, ou seja, se o alien 3 tiver selecionado o próximo alien ativo será o 4. Foi decidido implementar este sistema para facilitar a jogabilidade do jogador e para tornar o jogo uniforme.

1.2. Lock In

O **“lock in”** é um sistema para garantir que o jogador tomou as decisões corretas antes de avançar. Quando o jogador está a escolher membros da *crew*, os quartos em que estes aparecerão, os quartos dos *aliens*, etc. para avançar para o próximo estado têm que selecionar a opção **“lock in”**, desta forma o jogador tem sempre a possibilidade alterar as suas escolhas. Esta opção também é bastante útil para fazer debug do jogo.

1.3. Journey Tracker

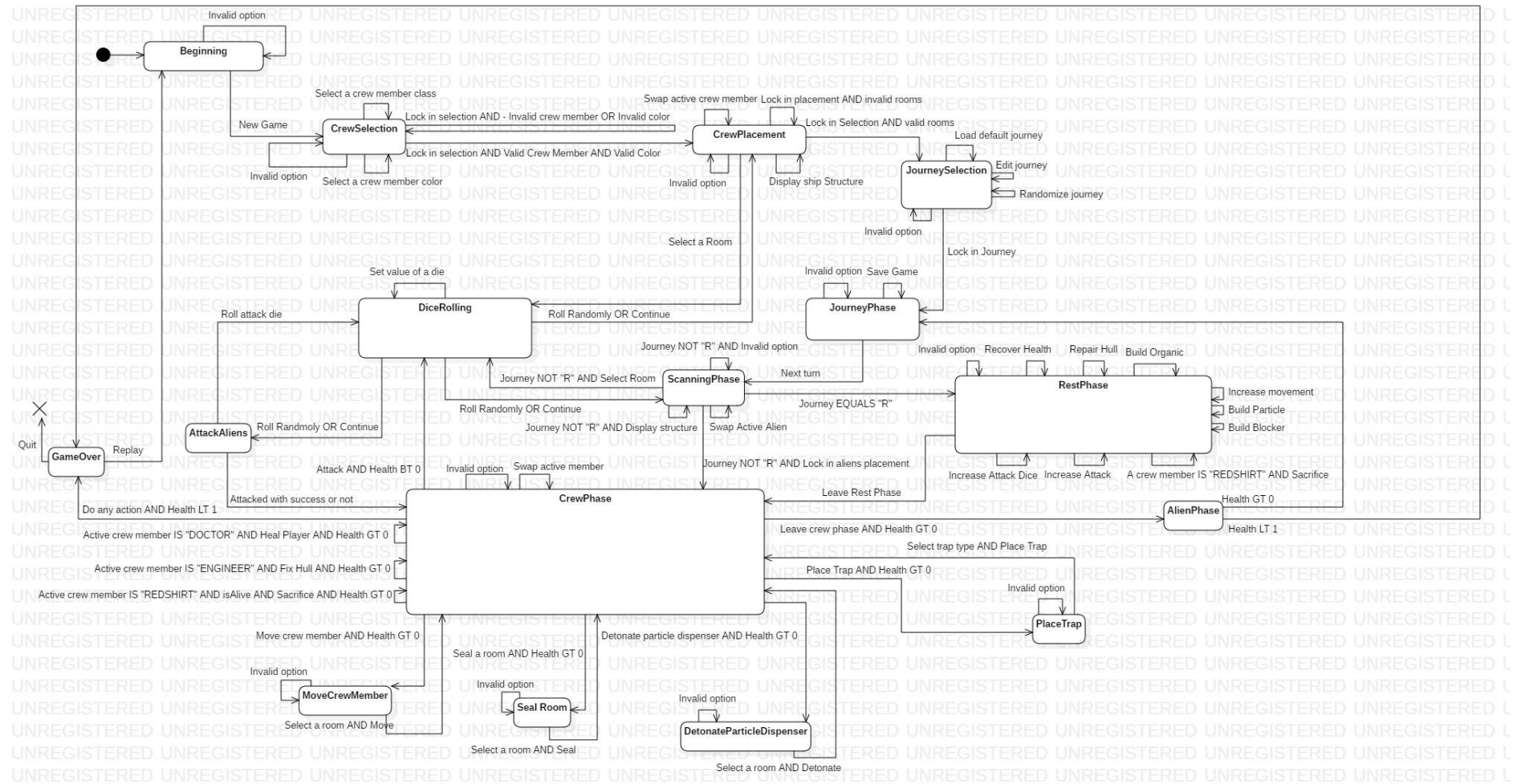
O “Journey Tracker” foi programado de modo a que o utilizador possa escolher o seu caminho, tornando então o jogo mais flexível. Para além de ter o *journey* por *default*, também são disponibilizadas as opções: **“Randomize Journey”** e **“Edit Journey”**. A opção **“Randomize Journey”** gera um caminho *random*, contudo, este *random* segue alguma regras para deixar o jogo balanceado, de modo a que não fique extremamente fácil ou difícil. A opção **“Edit Journey”** permite alterar um turno de cada vez, permitindo assim uma manipulação mais fácil.

1.4. Estados

Os estados implementados foram baseados na UI que o jogo iria ter. Antes de criar os estados os autores pensaram primeiro como queiram a UI do jogo e foram então desenvolvidos os estados. Esta forma de planejar os estados foi bastante útil e eficaz visto que os estados são baseados nas opções que o jogador pode fazer numa determinada fase do jogo e desta forma conseguimos logo ter uma noção de como iria ficar estruturado o jogo.

Também foi decidido implementar a sugestão de criar um estado para os dados, visto que várias ações do jogo vão necessitar dos dados, ter este estado torna a estrutura do jogo bastante mais flexível, fácil de utilizar e melhor de programar, visto que evita muitas redundâncias de código.

2. Máquina de estados



3. Classes

3.1. StateAdapter

A classe *StateAdapter* representa os estados do jogo. Esta contém os objetos: *game (DataGame)*, e métodos necessários para um funcionamento correto. O objetivo desta classe é permitir à lógica utilizador os estados, interligando os estados com a interface *IStates*.

3.2. DataGame

A classe *DataGame* representa os dados do jogo. Esta contém os objetos: *player (Player)*, *ship (Ship)*, *logs (List<String>)*, outros dados e a maioria dos métodos necessários para um funcionamento correto. O objetivo desta classe é organizar e gerir os dados do jogo.

3.3. Player

A classe *Player* representa os dados do jogador. Esta contém os objetos: da *Crew* (representados pela classe *CrewMember*), e outros dados e métodos necessários para um funcionamento correto. O objetivo desta classe é organizar e gerir os dados do jogador.

3.4. CrewMember

A classe *CrewMember* representa os dados de cada *crew member*. Esta classe é abstrata e as classes: *Captain*, *Commander*, *CommsOfficer*, *Doctor*, *Engineer*, *MoralOfficer*, *NavigationOfficer*, *RedShirt*, *ScienceOfficer*, *SecurityOfficer*, *ShuttlePilot*, *TransporterChief* estendem desta. Esta contém os objetos: *room (Room)*, e outros dados e métodos necessários para um funcionamento correto. O objetivo desta classe é representar e gerir os vários tipos de *crew members*.

3.5. Ship

A classe *Ship* representa os dados da nave. Esta contém os objetos: *rooms (HashMap<Integer><Room>)*, e outros dados e métodos necessários para um funcionamento correto. O objetivo desta classe é gerar a estrutura dos quartos (estática) na sua criação, organizar e gerir os dados dos quartos em geral.

3.6. Room

A classe *Room* representa os dados de cada quarto. Esta contém os objetos: *closestRoom s (List<Room>)*, *aliensInside (List<Alien>)*, *membersInside (List<CrewMember>)*, *trapInside (Trap)*, e outros dados e métodos necessários para o funcionamento correto do jogo. O objetivo desta classe é organizar e gerir os dados de cada quarto.

3.7. Trap

A classe *Trap* representa os dados das armadilhas. Esta classe é abstrata e as classes: *OrganicDetonator*, *ParticleDispenser* estendem desta. Esta contém dados

e métodos necessários para um funcionamento correto. O objetivo desta classe é organizar e gerir os dados de cada armadilha.

3.8. Alien

A classe ***Alien*** representa os dados de cada *alien*. Esta contém os objetos: ***room (Room)***, e outros dados e métodos necessários para o funcionamento correto do jogo. O objetivo desta classe é organizar e gerir os dados do *alien*.

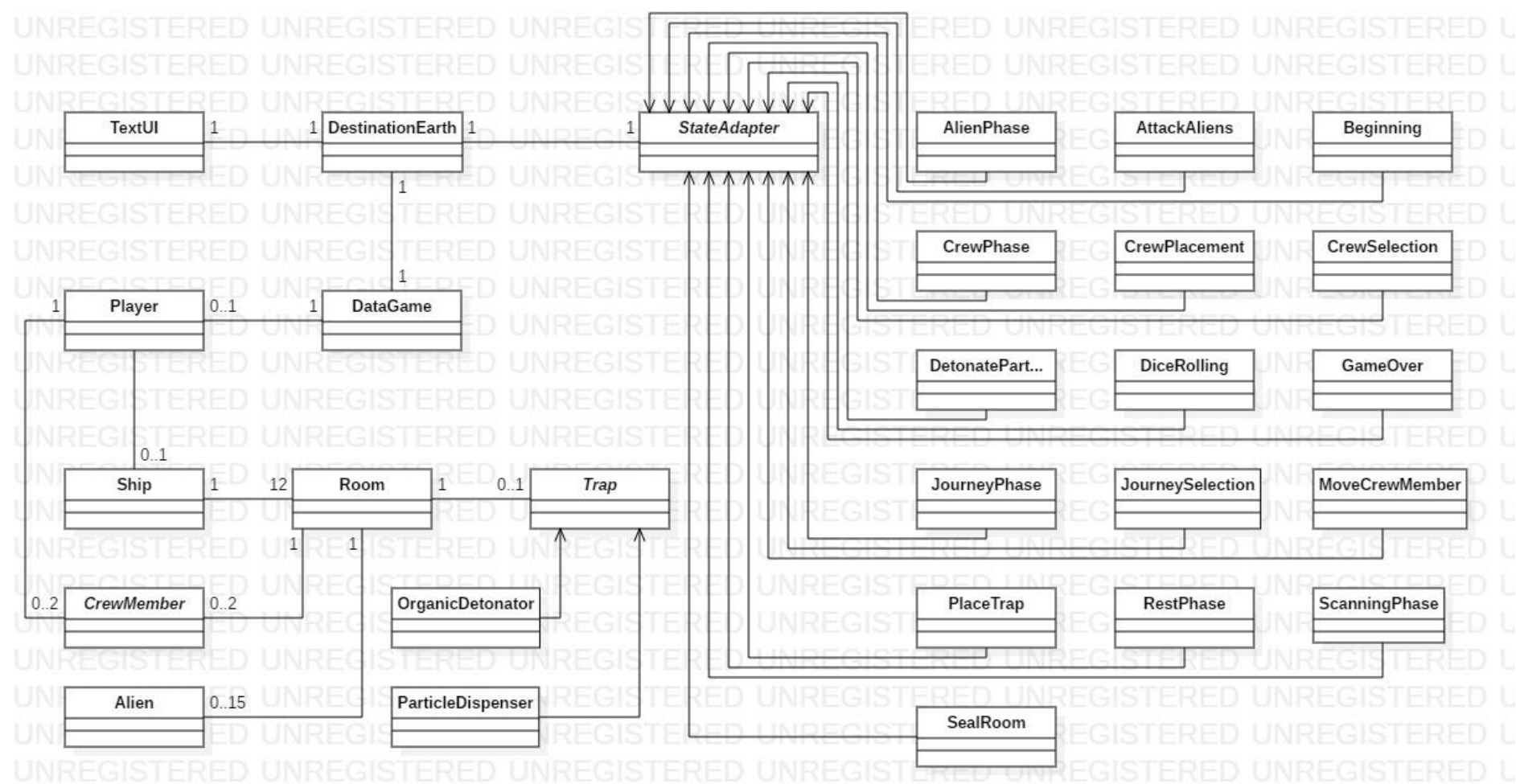
3.9. DestinationEarth

A classe ***DestinationEarth*** representa o jogo. Esta classe contém os objetos: ***dataGame (DataGame)***, ***state (IStates)***, e outros métodos, alguns necessários para um funcionamento correto, outros para a classe ***TextUI*** conseguir chamar métodos da classe ***DataGame*** de forma indireta. O objetivo desta classe é organizar o jogo.

3.10. TextUI

A classe ***TextUI*** representa a parte gráfica do jogo. Esta contém os objetos: ***game (DestinationEarth)***, outros dados necessários para um funcionamento correto e os métodos responsáveis por apresentar informação no ecrã. O objetivo desta classe é, apresentar as informações necessárias no ecrã e permitir que o utilizador interaja com o jogo.

4. Relacionamento de classes



5. Funcionalidades e regras

Feito	Requisito
✓	Save Game
✓	Load Game
✓	Arquitetura
✓	Máquina de estados
✓	Jogar novamente
✓	Opções de lançamento do dado
✓	<i>Alien Phase</i>
✓	Informações necessárias disponibilizadas ao jogador
✓	Estrutura e regras do jogo