Relatório de Projecto
Ano Lectivo 2016/2017

Engenharia e Desenvolvimento de Jogos Digitais
Paradigmas da Programação II
Docente: Prof. Luís Ferreira

Diogo Portela 13236
Pedro Silva 14281

Introdução

No âmbito da disciplina de PPII, foi-nos proposto fazer a transição de um jogo de tabuleiro para o formato digital. Após alguma ponderação, percebemos que queríamos investir o nosso tempo num projecto mais ambicioso e tentar ser o mais fieis possível ao Betrayal at the House on the Hill, jogo de tabuleiro publicado pela Avalon Hill.

O jogo de tabuleiro é baseado numa história de mistério e terror, na qual um grupo de três a seis pessoas exploram uma casa no topo de uma colina, e descobrem que esta está assombrada.

"Enter a new room and you might find something . . . or something might find you…"

Descrição do jogo

Em termos de peças no jogo original, temos:

- 44 cartas de quartos (utilizadas quando se abrem portas)
- 6 cartas de frente e verso para os jogadores
- 6 figuras de plástico
- 30 setas de plástico (usadas para marcar as stats atuais do jogador)
- 8 dados
- 1 Damage track (para controlar o progresso dos presságios e maldição)
- 13 cartas de presságio
- 22 cartas de equipamento / artefacto
- 45 cartas de evento
- 149 tokens para auxiliar alguns processos de jogo

Implementação

Para implementar o jogo em C, optámos por usar uma estrutura assente em listas ligadas para a execução do jogo e arrays para guardar a base de dados que contém as cartas jogáveis.

Em relação aos quartos temos a seguinte estrutura:

```
struct room
	{
		Vector2 position;
		int positionLenght;
		string name;
		RoomWall wall[WALL_NUMBER];
		EventPtr event;
		OmenPtr omen;
		BOOL isUsed;
		struct room *next;
	};
```

Tem um aspecto normal de uma *struct* usada para criar lista ligadas, com um apontador para uma estrutura igual, porém, contém também apontadores para um evento e/ou um omen (presságio), visto que a abertura de um quarto pode dar *trigger* a um o ou nenhum deles (neste caso ficam ambos a `NULL`). O booleano *isUsed* e o inteiro *positionLenght* são auxiliares para algumas das nossas funções e foram adicionados mais tarde. A *struct RoomWall* é usada para guardar os limites do quarto. Outra implementação possível para os quartos para além da lista ligada simples, seria conter um apontador para cada parede do quarto, e, caso uma parede tenha porta, ligar outro quarto através do apontador respectivo. Para guardar a posição no mapa de cada quarto, optámos por criar uma *struct* que é usada no decorrer do jogo:

```
struct vector2
	{
		int x;
		int y;
	};
```

Os jogadores são representados por doze cartas. Em vez de termos seis cartas de dupla face, em que cada face corresponde a uma variante da personagem, e visto que não estamos limitados pela existência de um peão para cada uma, implementámos espaço para doze personagens diferentes.

Para além disso usamos a seguinte estrutura:

```
struct Character
	{
		string name;
		int might, speed, sanity, inteligence;
		int playerNumber;
		Vector2 position;
		RoomPtr room;
		FloorPtr currentFloor;
		MinionPtr minions;
		ItemPtr items;
		HistoryPtr history;
		struct Character *next;
	};
```

Os inteiros, *might, speed, sanity* e *inteligence* são usadas para guardar as *stats* de cada jogador, sendo que estas vão sendo alteradas em paralelo com o decorrer do jogo. Seguidamente, o já referido Vector2 para representar a posição no ecrã de cada *character* bem como os apontadores para o respectivo quarto em que se encontra e o andar associado e a lista de equipamentos e *minions* que este possui. O *playerNumber* é um inteiro auxiliar que nos permite, de uma só forma, cobrir a necessidade de sinalizar se uma personagem foi escolhida, bem como qual dos jogadores ela representa. O apontador para a struct History que contém uma string, guardará uma descrição do movimento do jogador.

Para os *minions* e *items* usamos as semelhantes estruturas:

```
struct Minion
	{
		string name;
		int might_mod, speed_mod,
sanity_mod, intellect_mod;
		struct Minion *next;
	};
```

```
struct Item
	{
		string name, description;
		int might_mod, speed_mod,
sanity_mod, intellect_mod;
		struct Item *next;
	};
```

Os inteiros que integram estas estruturas servem para fazer a adição positiva ou negativa das *stats* do jogador em que os *minions* ou equipamentos interferem.

Para juntar estas structs, temos então, para a base de dados:

```
struct Card{
		Character characterList[MAX_CHARACTERS];
		unsigned int charCount;
		Event eventList[MAX_EVENTS];
		unsigned int eventCount;
		Item itemList[MAX_ITEMS];
		unsigned int itemCount;
		Minion minionList[MAX_MINIONS];
		unsigned int minionCount;
		Omen omenList[MAX_OMENS];
		unsigned int omenCount;
		Room roomList[MAX_ROOMS];
		unsigned int roomCount;	};
```

que contém os arrays dos possíveis objectos de jogo e contadores auxiliares, e, para o decorrer do jogo:

```
struct Master
	{
		Card cards;
		Map map;
		CharacterPtr characterList;
		int omenTrack;
	};
```

Assim, no decorrer do jogo, temos a base de dados guardada na variável *cards*, o mapa na *map* e o *omenTrack* guarda o estado da *damage track*. Guardamos ainda na estrutura a lista de personagens em jogo (que ja vimos que contém os outros objectos de jogo necessários).

As estruturas adicionais para gerir a estrutura de jogo criadas por conveniência podem ser observadas no ficheiro *Struct.h*.

Quanto a funções, temos as funções comuns de alocação de memória e criação de um novo objecto de jogo, bem como de inserção ordenada em listas, remoção, e outras funções auxiliares à nossa implementação do jogo.

Um exemplo notável é a função de abertura de quartos:

```
RoomPtr OpenRoom(MasterPtr master, FloorPtr floor, RoomPtr currentRoom, Direction
direction) {
		RoomWallPtr wallAux = FindWallDirection(currentRoom, direction);
		RoomPtr roomAux = NULL;
		RoomPtr roomRandom = NULL;
		Vector2 vectorAux;

		if (wallAux->WallType == DOOR)
		{
			vectorAux = currentRoom->position;
			if (direction == UP)
				vectorAux.y -= ROOM_SIZE;
			else if (direction == LEFT)
				vectorAux.x -= ROOM_SIZE * 2;
			else if (direction == DOWN)
				vectorAux.y += ROOM_SIZE;
			else if (direction == RIGHT)
				vectorAux.x += ROOM_SIZE * 2;

			roomRandom = RandomRoom(master);
			roomAux = InstanciateRoom(roomRandom, vectorAux);
			floor->roomList = AddRoomToList(floor->roomList, roomAux);
		}
		return roomAux;  };
```

A função recebe o *master*, o andar actual, o quarto actual e a direcção do movimento executado pelo jogador.

Depois de verificar o tipo de parede na direcção pretendida, cria uma instância de um quarto aleatoriamente retirado da base de dados e insere-o na lista dos quartos já existentes em jogo. No fim, devolve um apontador para o quarto que foi aberto.

Seguem as funções auxiliares à última:

```
RoomWallPtr FindWallDirection(RoomPtr room, Direction direction)
	{
		RoomWallPtr aux = NULL;
		if (direction == UP)
			aux = room->wall;
		else if (direction == LEFT)
			aux = &(room->wall[1]);
		else if (direction == DOWN)
			aux = &(room->wall[2]);
		else if (direction == RIGHT)
			aux = &(room->wall[3]);
		return aux;
	}

RoomPtr RandomRoom(MasterPtr master)
	{
		RoomPtr aux = NULL;
		int random = rand() % master->cards.roomCount;
		aux = &(master->cards.roomList[random]);

		if (aux->isUsed == TRUE)
			aux = RandomRoom(master);
		return aux;
	}



RoomPtr InstanciateRoom(RoomPtr room, Vector2 position)
	{
		RoomPtr aux = (RoomPtr)malloc(sizeof(Room));
		aux->event = room->event;
		strcpy(aux->name, strupr(room->name));
		aux->omen = room->omen;
		aux->position = position;
		aux->positionLenght = aux->position.x + aux->position.y;
		aux->next = NULL;
		room->isUsed = aux->isUsed =TRUE;
		BOOL i = CopyWalls(aux->wall, room->wall);

		if (i == TRUE)
			return aux;
		free(aux);
		return NULL;
	}
```

Outras funções relacionadas com a estruturação do jogo podem ser vistas no ficheiro *functions.c* do projecto da dll.

Quanto à parte gráfica, após tentarmos aplicar a livraria openGL para C, chegámos à conclusão que era preferível seguir para um estilo mais antigo e simples, e acabamos por usar caracteres ASCII.

O desenho de um quarto ficou então dez caracteres de largura e cinco de altura. ex:

```
##########
#        #
         +
#        #
###    ###
```

representando um quarto com uma parede em cima, janela do lado direito e duas portas, uma à esquerda e outra por baixo.

Para interagir com o utilizador, criámos estruturas que permitem que cada jogador, na sua vez de jogar, tenha impresso no ecrã a sua posição, mesmo que ela não esteja no mesmo andar. A câmera:

```
struct camera
{
	Vector2 MinBound;
	int minLenght;
	char viewPort[MAX_HEIGHT][MAX_WIDTH];
};
```

permite-nos obter este resultado, como que uma segunda *grid* que tem o seu próprio sistema de coordenadas.

Em termos de funções para a parte gráfica, usamos principalmente as seguintes funções para imprimir para o ecrã:

```
BOOL InsertSelectableText(string text, int x, int y, int currentSelected, int movingSelected, int movedX, int movedY, char(*drawingTable)[MAX_HEIGHT][MAX_WIDTH])
{
	Vector2 auxPos = ChangeVector2(x, y);
	if (currentSelected == movingSelected)
		auxPos = ChangeVector2(movedX, movedY);
	InsertLineInDrawingTable(*drawingTable, auxPos.x, auxPos.y, text);

	return TRUE;
}
```

usado principalmente para imprimir o menu (que optamos por ser linhas seleccionáveis em vez de esperar por um input de inteiros ou caracteres).

Para impressão do mapa bem como auxiliar a última função, usamos:

```
	BOOL	InsertLineInDrawingTable(char(*drawingTable)[MAX_HEIGHT][MAX_WIDTH],	int	x,
int y, string text)
	{
		int lenght = strlen(text);
		if (y > MAX_HEIGHT || y < 0)
			return FALSE;
		for (int i = 0; i < lenght; i++)
		{
			if (x < MAX_WIDTH && x >= 0)
			{
				(*drawingTable)[y][x] = text[i];
			}
			x++;
		}
		return TRUE;
	}
```

Nota-se, portanto que a impressão na consola é feita através de inserção num array bidimensional (*drawingTable*) que mais tarde será impressa por completo. É desta forma que a parte gráfica do jogo se processa, tentando aproximarmo-nos ao uso de *pixels*. Para outras funções da parte gráfica, ver o ficheiro graphics.c.

Para concluir, como foi proposto pelo professor, era suposto fazermos alguma adição ou alteração ao jogo original. A nossa opção para este ponto foi dar ao utilizador ferramentas para poder alterar toda a base de dados do jogo, criar as suas próprias cartas, podendo gravar vários ficheiros de base de dados e carregar a que pretende usar, antes de cada jogo. Assim, o utilizador tem toda a liberdade para jogar o jogo da forma que achar mais divertida.

Conclusão

Após a realização do trabalho verificamos que foi um pouco ambicioso demais, visto que o jogo de tabuleiro é bastante complexo por si só. Para agravar a situação, passámos uma semana a tentar implementar os gráficos em openGL, opção que acabou por ser completamente descartada. Porém, a parte mais importante para o objectivo do trabalho, a estrutura de dados e o seu funcionamento, foi muito bem conseguida.

Em suma, o projecto cumpre o objectivo. Não está, porém, em termos de interação com o utilizador, da forma que gostaríamos que estivesse. Contudo, se posteriormente o quisermos completar, alguns dias será o suficiente. Percebemos também que a linguagem C é extremamente versátil e que podiamos implementar o

jogo de inúmeras maneiras. Se recomeçado, teríamos certamente diferentes abordagens para algumas partes do trabalho.

Possíveis melhorias para o futuro

- Melhorar interação com os jogadores.
- Melhorar os gráficos com auxílio da livraria openGL.
- Separar e bloquear a base de dados original, de forma a não poder ser alterada.
- Adicionar som temático ao jogo.