Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Ciências da Computação
INE5412 – Sistemas Operacionais I
Professores: Giovani Gracioli e Márcio Bastos Castro

Alunos: Rafael Begnini de Castilhos – 20205642
Rafael Moresco Vieira – 19200435

Projeto Final Pacman

Florianópolis, 19 de Julho de 2022

Índice

Resumo

Neste relatório é apresentado o projeto e desenvolvimento do trabalho final da unidade curricular INE5412, Sistemas Operacionais I, através de diagramas UML e imagens, explicando qual foi o raciocínio para o desenvolvimento do código.

1. Introdução

O projeto Pacman tem como objetivo pôr em prática e avaliar tudo adquirido ao longo da disciplina, não só em conhecimento, mas em software, dependendo da biblioteca que simula o funcionamento de um Sistema Operacional desenvolvida ao longo do semestre. Foram disponibilizados para a produção do Pacman todas as *sprites* e imagens, já com uma função inicial para demonstrar o carregamento da parte gráfica, e um sistema já implementado para identificar a entrada de dados do teclado.

2. Dependências

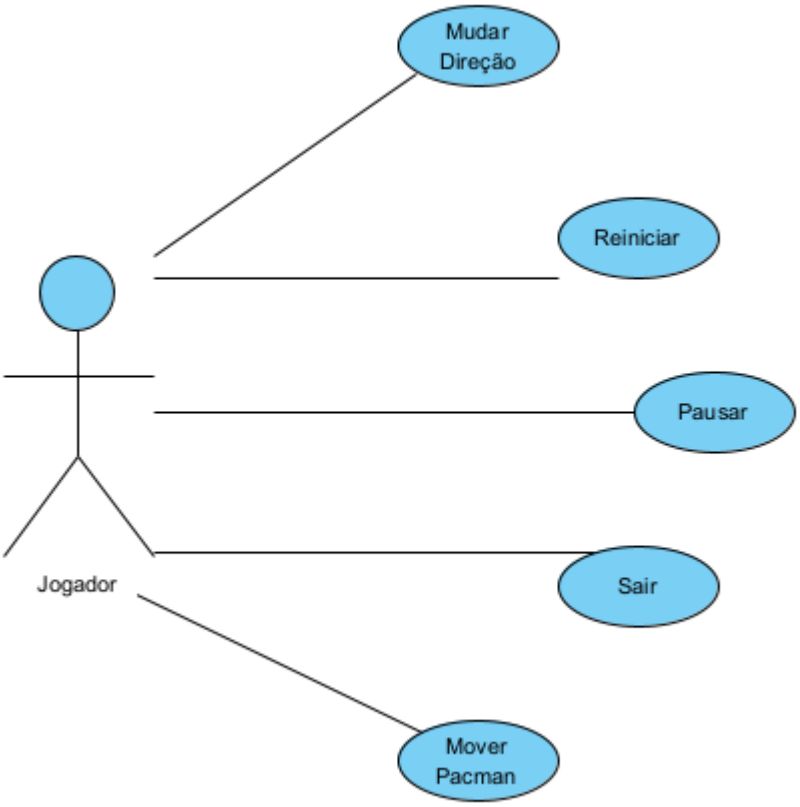
Além da biblioteca “os” desenvolvida ao longo do semestre, outras duas bibliotecas foram adicionadas para o desenvolvimento do projeto. A primeira é a *Simple and Fast Multimedia Library* (SFML), indicada pelos professores, uma biblioteca feita para dar uma interface simples para diversos componentes do computador, com o objetivo o desenvolvimento de aplicações multimídia e jogos. A segunda é a *libpng*, uma biblioteca específica para tratamento de arquivos *PNG*, sendo a biblioteca oficial do formato *PNG*, dando suporte para quase todas as *features* do PNG. As bibliotecas podem ser adquiridas através dos comandos *sudo apt-get install libpng-dev* e *sudo apt-get install libsfml-dev*.

3. Projeto

Para o desenvolvimento do projeto, foi utilizado as práticas e normas aprendidas na unidade curricular Engenharia de Software (INE5417), desenvolvendo diagramas UML através da ferramenta *Visual Paradigm*. Os passos iniciais nesse tipo de desenvolvimento é avaliar quais são os casos de uso do programa, e quais as classes necessárias para modelar o comportamento desejado no domínio do problema e no domínio da solução.

3.1 Casos de Uso

Os casos de uso representam como um ator interage com o programa desenvolvido. Neste caso, o ator representa o jogador. Segue abaixo o diagrama dos casos de uso.



O jogador interage com o programa através do uso das teclas de direção e as teclas “P”, “Q” e “R”, algo que é especificado pela descrição do projeto definida pelos professores. Com essas teclas, o jogador pode, respectivamente, mudar a direção do Pacman, pausar a partida, sair da partida e reiniciar a partida quando ela termina. Apesar do movimento do Pacman ser automático quando uma direção é definida foi decidido colocar como um caso de uso por duas razões, ele pode ser “cancelado” com o uso constante das teclas de mudança de direção, e para posteriormente modelar o diagrama de sequência para o caso de uso.

3.2 Diagrama de Classe

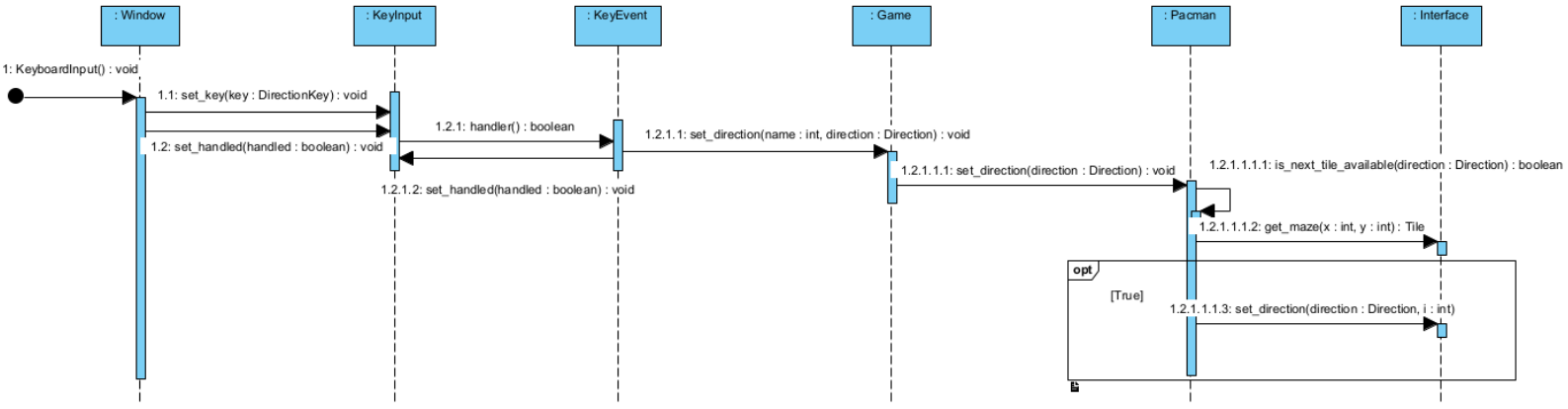
Com os casos de uso definidos, a próxima etapa do projeto é definir os diagramas de classes. Na área do domínio do problema, foram planejadas cinco classes. São elas: *Char*, *Game*, *Interface*, *Pacman*, *Ghost*. A classe *Char* representa todas as personagens do jogo, possuindo posição, direção e nome como atributos principais, e *setters*, *getters*, movimento e atualização da posição como métodos principais. As classes *Pacman* e *Ghost* são herdeiras dessa classe. A classe *Pacman* só adiciona um método de *reset* a classe genérica, enquanto a classe *Ghost* adiciona diversos métodos e atributos para o controle do comportamento dos fantasmas. A classe *Game* é responsável pelo controle de diversas lógicas do jogo, como identificar os estados do jogo, e atualizar os estados de elementos das classes *Interface*, *Pacman* e *Ghost*. A classe *Interface*, apesar de possuir esse nome, não é a classe de

interação do jogador com o programa, mas sim a classe responsável pelo labirinto do jogo, possuindo a matriz do labirinto e armazenando a posição das personagens. Essa classe também guarda informações adicionais como a pontuação e a quantidade de vidas que o jogador possui.

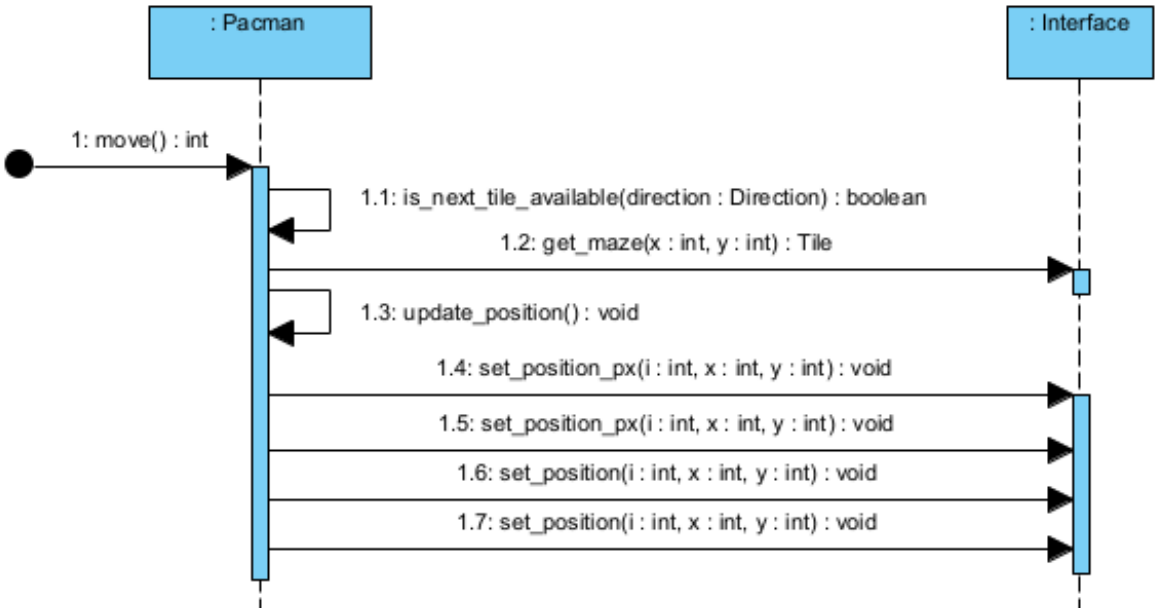
Já no domínio da solução, encontram-se outras oito classes, com as três principais sendo: *KeyInput*, *KeyEvent* e *Window*. As outras classes são classes da biblioteca SFML, usadas para compor a classe *Window*. A classe *Window* representa a interação entre o jogador e o programa, efetivamente sendo o ator no diagrama de casos de uso. Através dela, são desenhadas as *textures* e *sprites*, além de possuir a entrada do teclado. As informações do teclado recebidas pela classe *Window são tratadas pelas classes KeyInput* e *KeyEvent*, sendo a primeira onde a informação de qual tecla foi pressionada é guardada, mandando para a segunda que trata essa informação, mandando comandos para a classe *Game*. O diagrama de classes se encontra anexado ao fim do documento.

3.3 Diagramas de Sequência

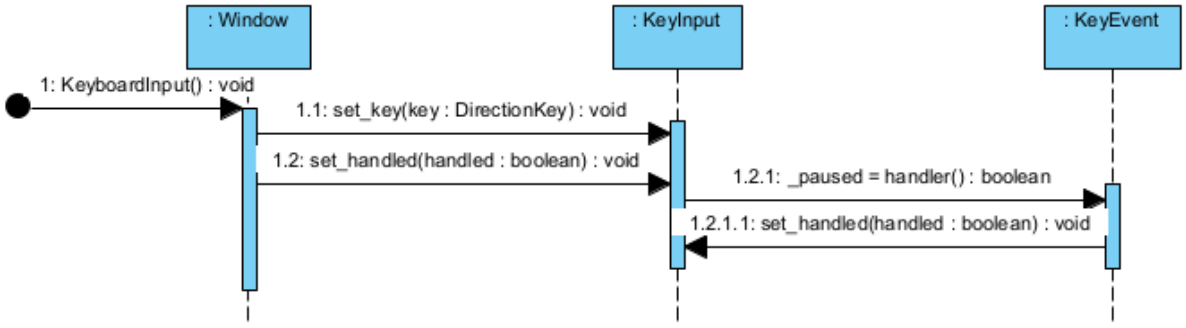
Para refinar os casos de uso, foram criados diagramas de sequência para demonstrar o funcionamento e a interação de classes que esse caso de uso demanda. Os mais importantes são os diagramas Mudar Direção e Mover Pacman.



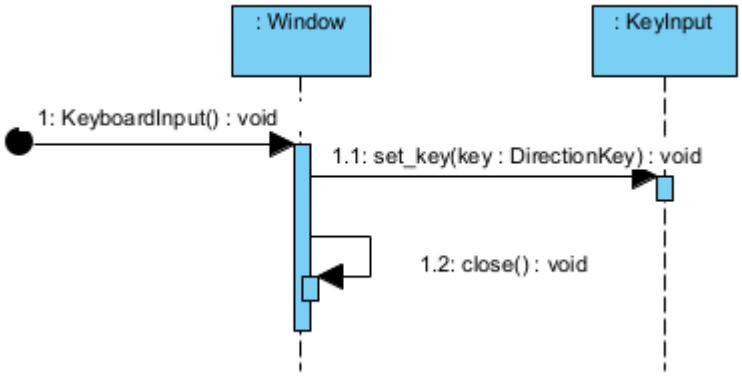
Primeiramente, se encontra o diagrama Mudar Direção. Com ele pode-se ver a interação vindo do usuário através da chamada do método *KeyboardInput()*, na classe *Window*, que é tratada pelas classes *KeyInput* e *KeyEvent*. *Window* inicialmente define a tecla pressionada como não tratada depois de mandar para *KeyInput*, e ao tratar dessa tecla, *KeyEvent* informa *KeyInput*. O tratamento da tecla ocorre chamando o método *set_direction()* da classe *Game*, informando qual tecla foi pressionada. *Game*, por sua vez, repassa essa informação para a classe *Pacman*, que verifica se a direção requisitada é válida, verificando com a posição armazenada na classe *Interface*. Se a direção for válida, *Pacman* define ela como sua nova direção, informando a *Interface*.



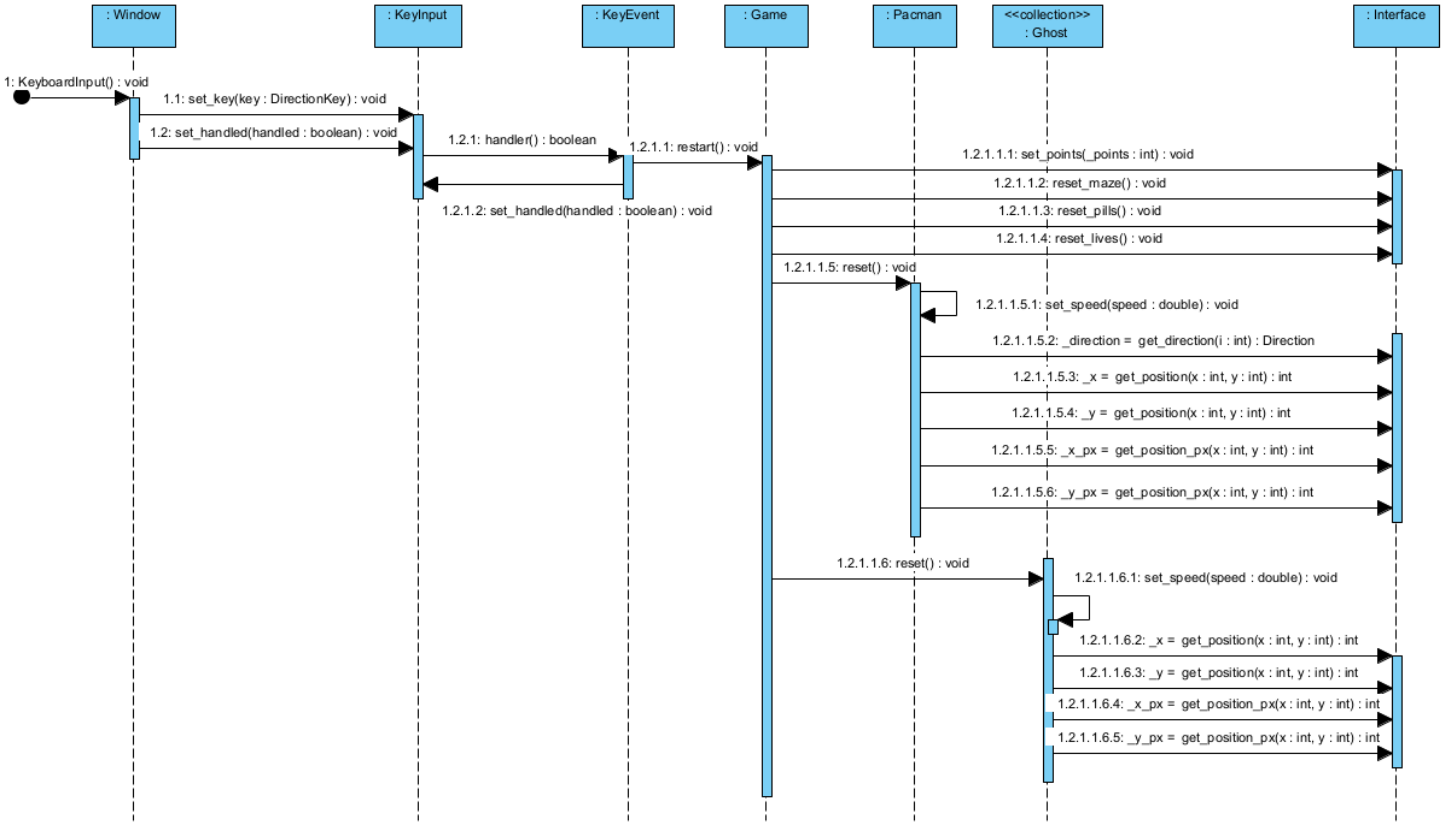
Com o diagrama Mover Pacman, é visto que a interação não vem do jogador através da classe *Window*, mas sim da lógica automática do programa. Quando um ciclo ocorre, o método *move()* da classe *Pacman* automaticamente é chamado, e em seguida a classe verifica se a próxima posição no labirinto é válida. Para isso, é chamado o método *get_maze()* da classe *Interface*. Mesmo se a posição for inválida, é feito uma atualização da posição da classe *Pacman*, mandando as informações da nova posição (ou a mesma caso inválida) para a *Interface*. Segue o diagrama de sequência dos outros casos de uso.



Com o diagrama de sequência Pausar, observa-se que seu funcionamento é similar ao tratamento de tecla do Mudar Direção, contudo direto na classe *KeyEvent* a variável que indica que o jogo está pausado é definida, bloqueando todas as ações até ela ser pressionada novamente.



Com o diagrama Sair, percebe-se um tratamento de tecla muito mais simples. Como a classe *Window* é responsável pela janela de interação, ela diretamente pode terminar o programa, terminando sua thread.



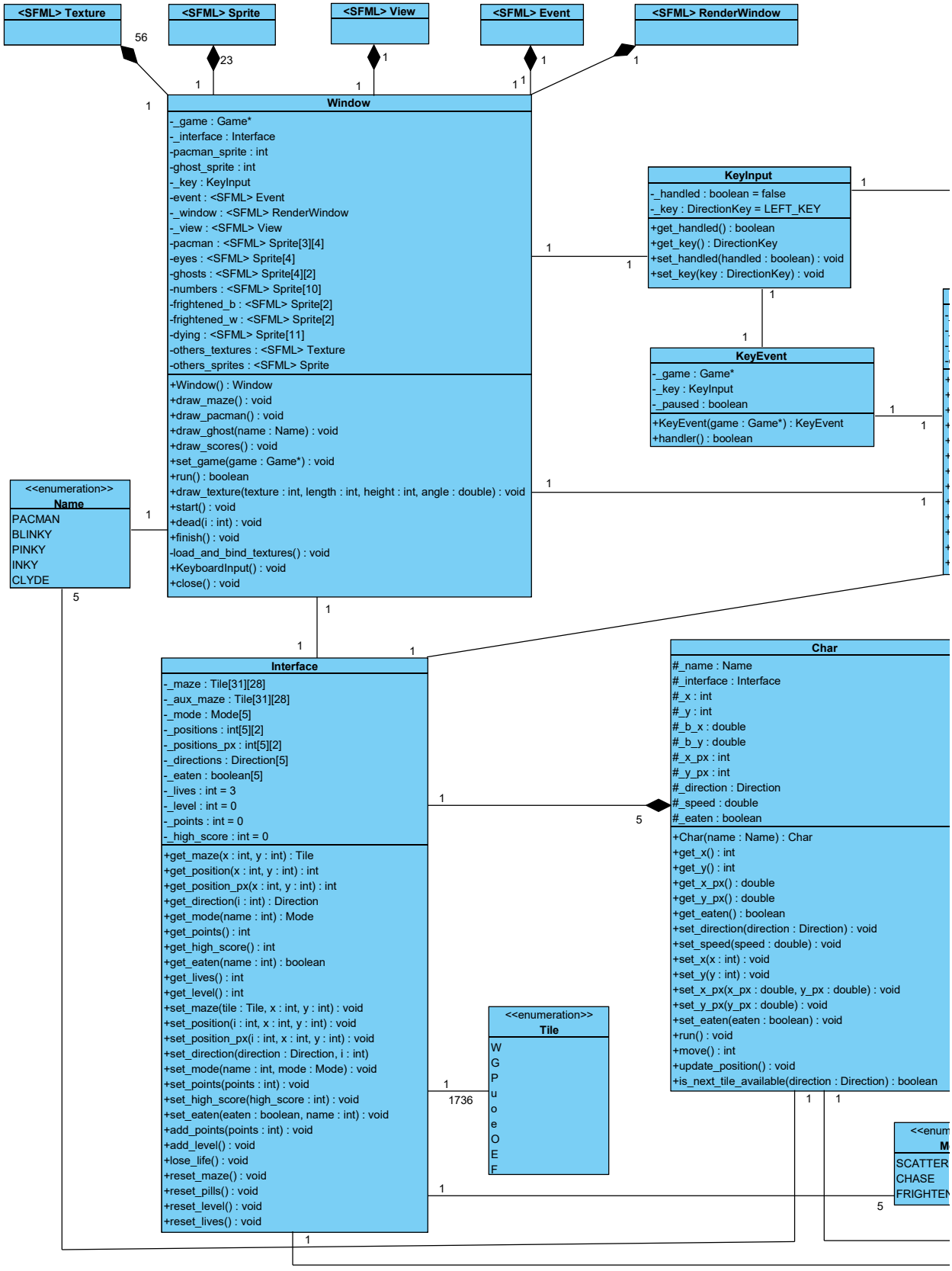
No diagrama Reiniciar, encontra-se o tratamento normal da entrada do teclado, como no diagrama Mudar Direção. A tecla é tratada com a classe *Game* reiniciando as informações guardadas na *Interface*, e em seguida as personagens, reiniciando a classe *Pacman* e as quatro classes *Ghost* instanciadas. Essas classes reiniciam de forma similar, a diferença sendo que a classe *Pacman* reinicia com a posição definida na classe Interface.

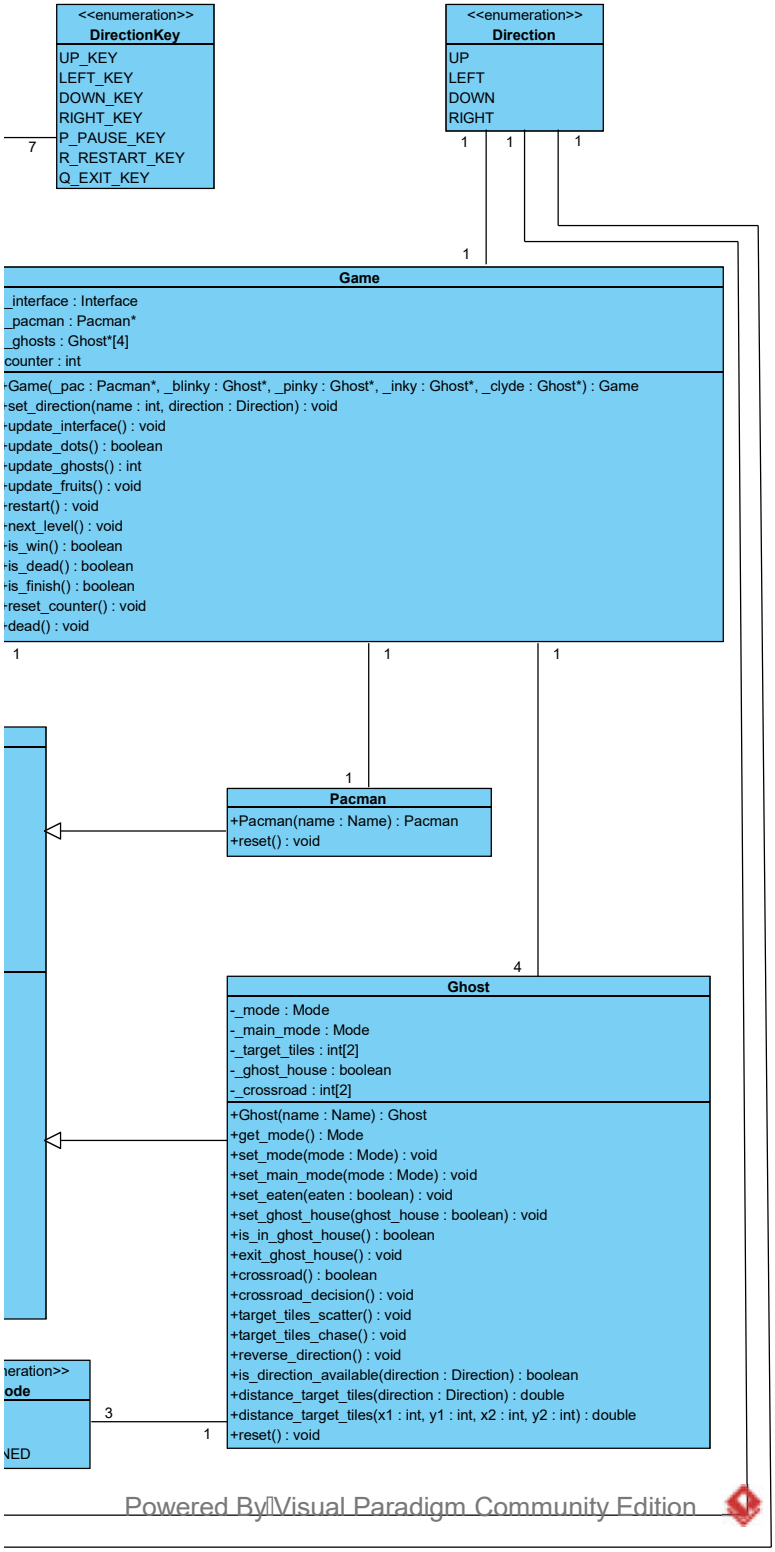
4. Conclusão

Com o projeto Pacman, foi possível pôr em prática de uma maneira divertida todo o trabalho desenvolvido ao longo do semestre. A biblioteca produzida em sala de aula deixa de ser apenas uma atividade avaliativa isolada, e passa a ser parte de um processo construtivo que necessita de muita dedicação. Além disso, foi possível utilizar os conhecimentos de outras disciplinas vistas até agora no curso, demonstrando um crescimento como cientistas da computação.

5. Referências

HALLER, J.; HANSSON, H. V.; MOREIRA, A. ***SFML Game Development,*** Editora Packet, 2013


<SFML> Texture
<SFML> Sprite
<SFML> View
<SFML> Event
<SFML> RenderWindow
56
23
1
1
1
1
1
1
1
1
Window
-_game : Game*
-_interface : Interface
-pacman_sprite : int
-ghost_sprite : int
-_key : KeyInput
-event : <SFML> Event
-_window : <SFML> RenderWindow
-_view : <SFML> View
-pacman : <SFML> Sprite[3][4]
-eyes : <SFML> Sprite[4]
-ghosts : <SFML> Sprite[4][2]
-numbers : <SFML> Sprite[10]
-frightened_b : <SFML> Sprite[2]
-frightened_w : <SFML> Sprite[2]
-dying : <SFML> Sprite[11]
-others_textures : <SFML> Texture
-others_sprites : <SFML> Sprite
+Window() : Window
+draw_maze() : void
+draw_pacman() : void
+draw_ghost(name : Name) : void
+draw_scores() : void
+set_game(game : Game*) : void
+run() : boolean
+draw_texture(texture : int, length : int, height : int, angle : double) : void
+start() : void
+dead(i : int) : void
+finish() : void
-load_and_bind_textures() : void
+KeyboardInput() : void
+close() : void
KeyInput
-_handled : boolean = false
-_key : DirectionKey = LEFT_KEY
+get_handled() : boolean
+get_key() : DirectionKey
+set_handled(handled : boolean) : void
+set_key(key : DirectionKey) : void
KeyEvent
-_game : Game*
-_key : KeyInput
-_paused : boolean
+KeyEvent(game : Game*) : KeyEvent
+handler() : boolean
<<enumeration>>
Name
PACMAN
BLINKY
PINKY
INKY
CLYDE
5
Interface
-_maze : Tile[31][28]
-_aux_maze : Tile[31][28]
-_mode : Mode[5]
-_positions : int[5][2]
-_positions_px : int[5][2]
-_directions : Direction[5]
-_eaten : boolean[5]
-_lives : int = 3
-_level : int = 0
-_points : int = 0
-_high_score : int = 0
+get_maze(x : int, y : int) : Tile
+get_position(x : int, y : int) : int
+get_position_px(x : int, y : int) : int
+get_direction(i : int) : Direction
+get_mode(name : int) : Mode
+get_points() : int
+get_high_score() : int
+get_eaten(name : int) : boolean
+get_lives() : int
+get_level() : int
+set_maze(tile : Tile, x : int, y : int) : void
+set_position(i : int, x : int, y : int) : void
+set_position_px(i : int, x : int, y : int) : void
+set_direction(direction : Direction, i : int)
+set_mode(name : int, mode : Mode) : void
+set_points(points : int) : void
+set_high_score(high_score : int) : void
+set_eaten(eaten : boolean, name : int) : void
+add_points(points : int) : void
+add_level() : void
+lose_life() : void
+reset_maze() : void
+reset_pills() : void
+reset_level() : void
+reset_lives() : void
Char
#_name : Name
#_interface : Interface
#_x : int
#_y : int
#_b_x : double
#_b_y : double
#_x_px : int
#_y_px : int
#_direction : Direction
#_speed : double
#_eaten : boolean
+Char(name : Name) : Char
+get_x() : int
+get_y() : int
+get_x_px() : double
+get_y_px() : double
+get_eaten() : boolean
+set_direction(direction : Direction) : void
+set_speed(speed : double) : void
+set_x(x : int) : void
+set_y(y : int) : void
+set_x_px(x_px : double, y_px : double) : void
+set_y_px(y_px : double) : void
+set_eaten(eaten : boolean) : void
+run() : void
+move() : int
+update_position() : void
+is_next_tile_available(direction : Direction) : boolean
5
1736
<<enumeration>>
Tile
W
G
P
u
o
e
O
E
F
<<enum
M
SCATTER
CHASE
FRIGHTEN
5


<<enumeration>>
DirectionKey
UP_KEY
LEFT_KEY
DOWN_KEY
RIGHT_KEY
P_PAUSE_KEY
R_RESTART_KEY
Q_EXIT_KEY
7
<<enumeration>>
Direction
UP
LEFT
DOWN
RIGHT
1
1
1
1
Game
_interface : Interface
_pacman : Pacman*
_ghosts : Ghost*[4]
counter : int
+Game(_pac : Pacman*, _blinky : Ghost*, _pinky : Ghost*, _inky : Ghost*, _clyde : Ghost*) : Game
+set_direction(name : int, direction : Direction) : void
+update_interface() : void
+update_dots() : boolean
+update_ghosts() : int
+update_fruits() : void
+restart() : void
+next_level() : void
+is_win() : boolean
+is_dead() : boolean
+is_finish() : boolean
+reset_counter() : void
+dead() : void
1
1
1
1
Pacman
+Pacman(name : Name) : Pacman
+reset() : void
4
Ghost
-_mode : Mode
-_main_mode : Mode
-_target_tiles : int[2]
-_ghost_house : boolean
-_crossroad : int[2]
+Ghost(name : Name) : Ghost
+get_mode() : Mode
+set_mode(mode : Mode) : void
+set_main_mode(mode : Mode) : void
+set_eaten(eaten : boolean) : void
+set_ghost_house(ghost_house : boolean) : void
+is_in_ghost_house() : boolean
+exit_ghost_house() : void
+crossroad() : boolean
+crossroad_decision() : void
+target_tiles_scatter() : void
+target_tiles_chase() : void
+reverse_direction() : void
+is_direction_available(direction : Direction) : boolean
+distance_target_tiles(direction : Direction) : double
+distance_target_tiles(x1 : int, y1 : int, x2 : int, y2 : int) : double
+reset() : void
3
1
Powered By Visual Paradigm Community Edition