Laboratórios de Computadores – MIEIC

2015/2016

Signor Mario

Turma 6 – Grupo 15

Andreia Rodrigues up201404691
Eduardo Leite gei12068

Indice

Contexto

O projeto desenvolvido pelo grupo, "Signor Mario", baseia-se no jogo *"Super Mario"* da Nintendo. Um jogo 2D do estilo *platformer* com um sistema de níveis.

O objetivo do jogo é passar pelos diferentes níveis, evitando obstáculos e coletando moedas no menor tempo possível, sem que os *Health Points* da personagem cheguem a zero.

No nosso projeto existem dois elementos capazes de retirar HP ao jogador, picos e esqueletos, sendo que estes devem ser evitados pelo jogador. Sempre que exista contacto entre o jogador e estes obstáculos, um ponto de vida será retirado ao jogador e será aplicada uma força sobre ele (*knockback)* que o empurrará para a direção contrária.

Temos também implementadas caixas e plataformas sobre as quais o jogador se consegue mover, sendo que um tipo especial de caixas podem ser partidas com o uso do cursor. Tudo isto é controlado através de um sistema de colisões.

Um outro obstáculo presente no jogo é o facto de existir um tempo limite para a conclusão do nível, o jogador tem apenas dois minutos para atravessar todo o mapa e chegar à bandeira que marca o final do nível.

O jogo guarda também as 5 melhores pontuações, através do uso de ficheiros, sendo que o jogador é informado caso consiga uma pontuação melhor do que as anteriores, em qualquer caso a pontuação final é mostrada. A pontuação final é calculada através de uma combinação entre o número de moedas coletadas e o tempo que se levou a terminar o nível.

Estado do Projeto

Todos os objetivos foram atingidos, à exceção do uso do UART.

Device	What for	Interrupts? / Pooling?
Timer	Game time / Drawing Cicle	Interrupts
Placa Gráfica	Draw, filter, flip, animate sprites, double buffering, font system	Interrupts
Teclado	Controls the character and the interaction between menus	Interrupts
Rato	Select menu options, break specific blocks	Interrupts
RTC	Read the date for the high scores list	Pooling

Placa Gráfica:

- Funciona perfeitamente
- Desenha bitmaps
- Roda bitmaps
- Filtra cores
- Suporta animações de personagens e deteção de colisões entre objetos
- Tem font system
- Modo de vídeo 117, o jogo deve ser corrido em full screen
- É utilizado double buffer para desenho dos bitmaps

Implementado em VBE.c, VBE.h, Graphics.c e Graphics.h. Funções relativas à placa gráfica são chamadas em GameLoop.c, onde se desenvolve todo o programa.

Teclado:

- Controla a personagem de uma forma muito fluída, tanto nos saltos como no movimento horizontal
- É capaz de decidir o que fazer com o Input, em especial da tecla ESC de acordo com o estado do

jogo

Implementado em Keyboard.c e Keyboard.h. Interrupções detetadas em GameLoop.c, tanto no estado do menu principal, como no próprio estado de jogo.

Rato:

- Navega de forma fluida em qualquer contexto, dentro dos limites do ecrã
- É capaz de detetar cliques no botões to menu
- Consegue interagir com elementos do nível

Implementado em Mouse.c e Mouse.h. Interrupções detetadas em GameLoop.c, tanto no estado do menu principal, como no próprio estado de jogo.

RTC:

- É capaz de aceder à data, necessária para o controlo de Scores

Implementado em RTC.c e RTC.h. Funções chamadas em GameLoop.c, para utilização dos dados em ScoreManager.c.

Câmara:

- É capaz de controlar a posição de todos os elementos do jogo
- Se necessário, é possível navegar pelo mapa utilizando esta classe (util para debug)

Implementada em Camara.c e Camara.h. Funções chamadas em GameLoop.c, onde é atualizada a posição de todos os objetos.

Máquinas de estado:

- Controlam o movimento da personagem

- Controlam o estado do menu

Implementadas em GameLoop.c, em Character.c e em Character.h

Colisões:

- Implementamos um sistema de colisões complexo, que apesar de não ter a perfeição do jogo original, tem em conta 8 tipos de colisões diferentes (os 4 cantos e as 4 superfícies), sendo capaz de diferenciar estas colisões e fazer os cálculos de acordo com a situação
- Funciona em conjunto com o sistema de velocidades e sabe reconhecer se o jogador está ou não em cima de uma superfície sobre a qual consiga saltar
- Também é capaz de diferenciar um salto em cima de um esqueleto (fatal para o esqueleto) de uma colisão de lado (fatal para a personagem) e de agir de acordo com o sucedido lançando o jogador com *knockbacks* diferentes

Implementadas em Boxes, Enemy, Spikes, Platform, e Coins (.c / .h). Funções relativas ás colisões são chamadas em GameLoop.c.

Física:

- O programa utiliza gravidade e um sistema de velocidades 2D
- Tem sempre em conta se o jogador está ou não no chão
- Sabe distinguir ações vindas do *input* de ações vindas de colisões, retirando o controlo da personagem ao utilizador sempre que necessário

Implementadas em Character.c e Character.h.

Ficheiros:

- Lê de um ficheiro
- Escreve para um ficheiro

Implementados em ScoreManager.c e ScoreManager.h.

Known bugs:

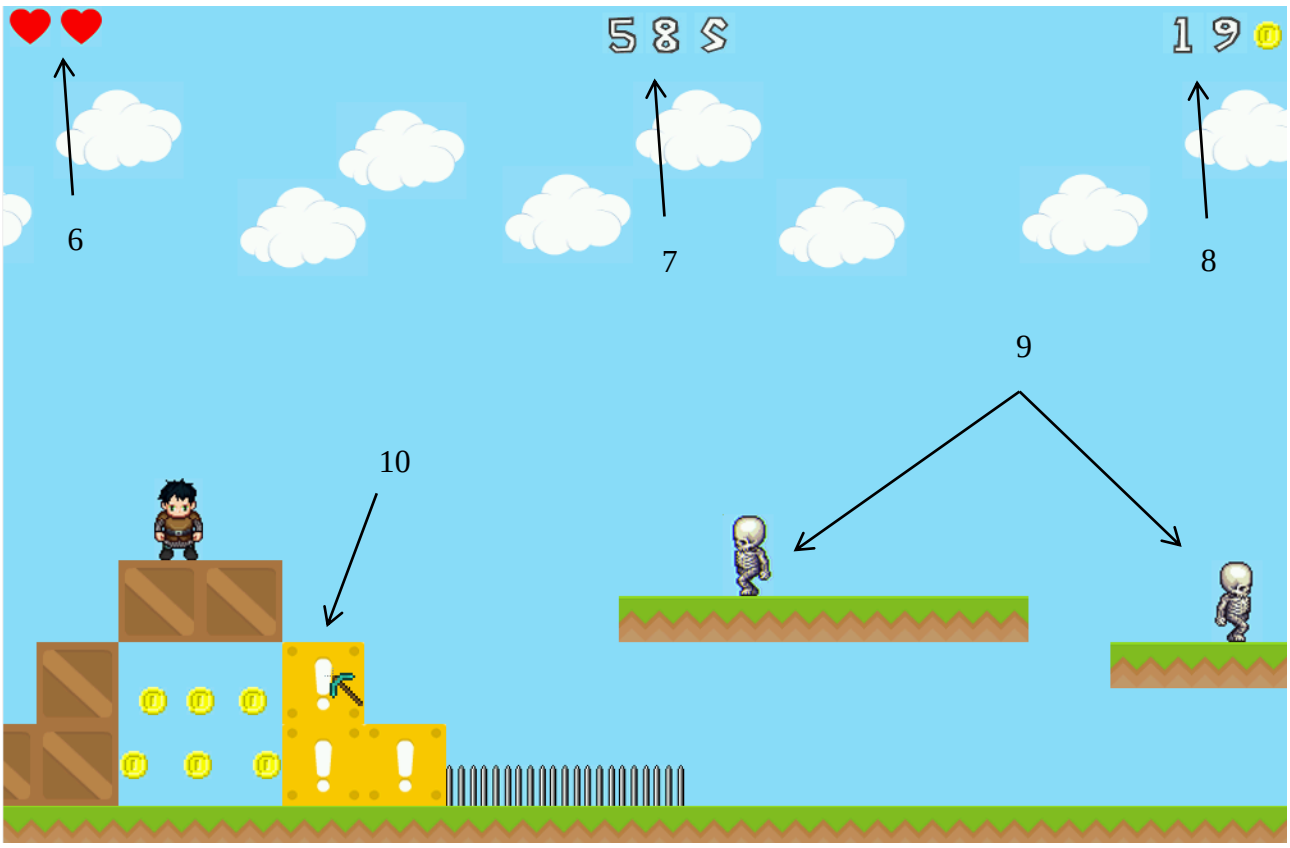
- Por vezes (muito raramente) as imagens não carregam corretamente
- O quarto *score* por vezes não é desenhado corretamente, de forma inexplicável, ele entra num ciclo for de desenho com o valor correto junto com todos os outros valores, mas o seu valor não é desenhado, seja ele qual for. (enfâse no por vezes)

Nota Pessoal:

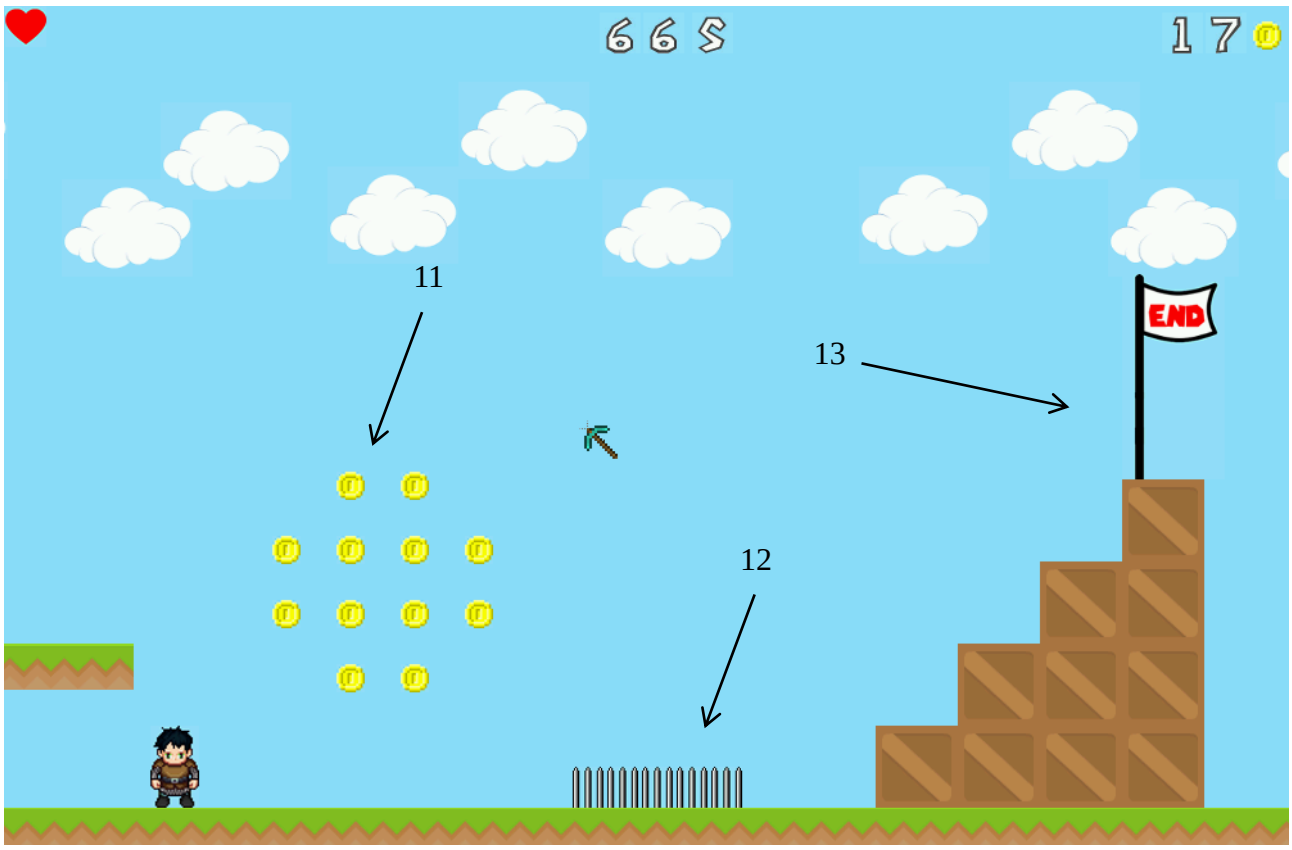
Contudo, como em qualquer outro jogo, há sempre espaço para melhorar o conteúdo existente ou para adicionar novos elementos. Caso o professor decida que tem um uso para o nosso jogo, ou sobre um tempo extra nas férias, revisitar o "*Signor Mario*" é uma opção que ainda temos em consideração.

Instruções de Utilização

1. Começa o jogo
2. Mostra os controlos
3. Mostra os high scores
4. Mostra os créditos
5. Sai do jogo



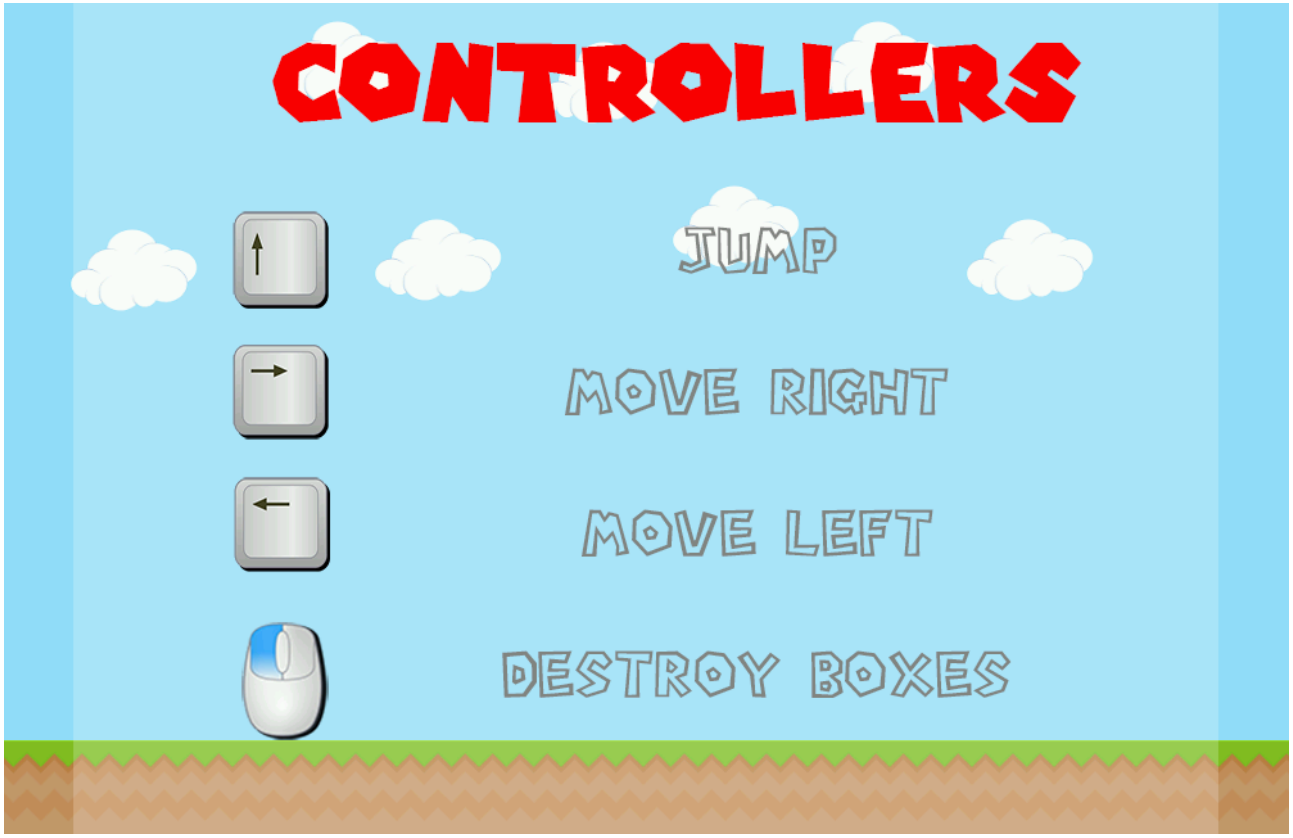
6. Health Points restantes
7. Tempo de jogo passado
8. Moedas coletadas
9. Inimigos
10. Caixas destrutíveis



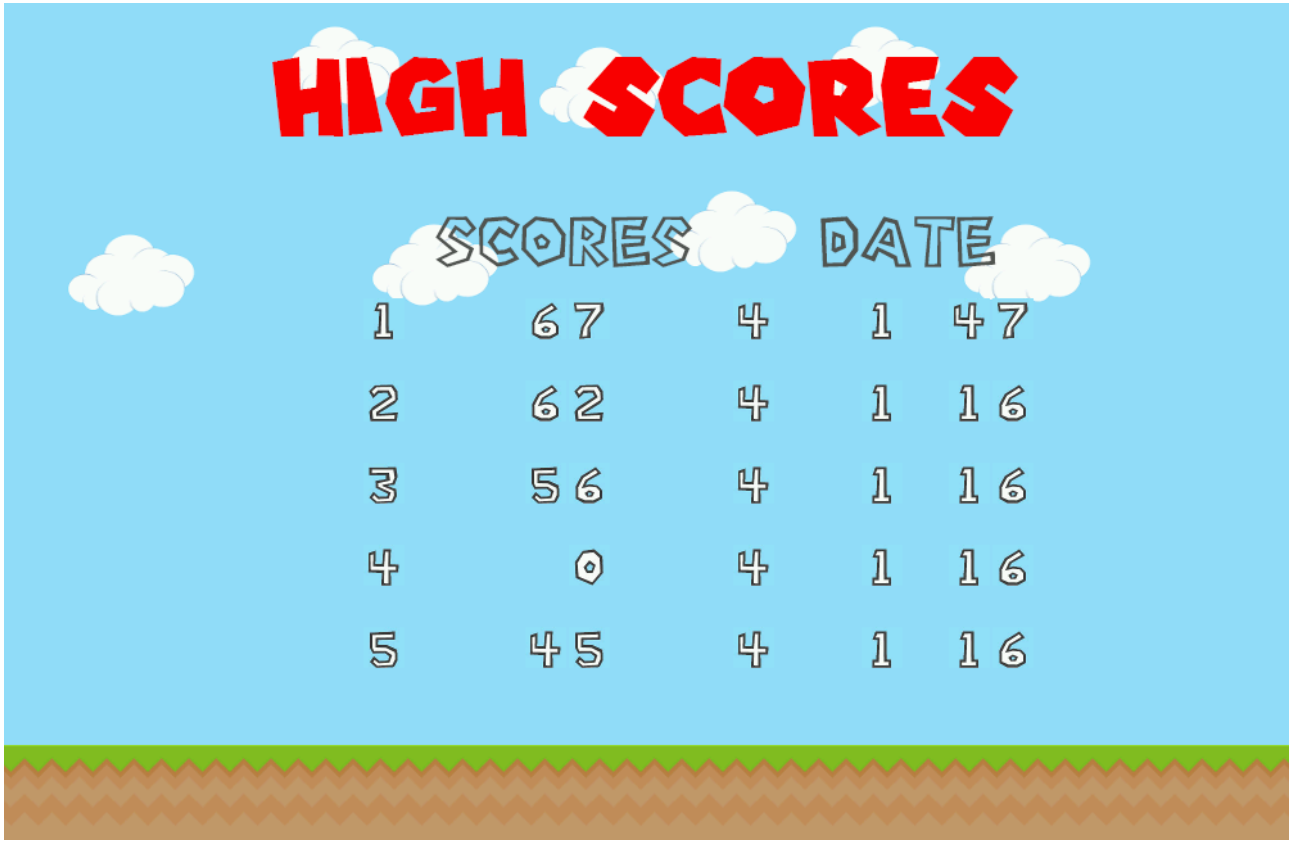
11. Moedas para coletar

12. Picos

13. Bandeira, assinala o final do nível

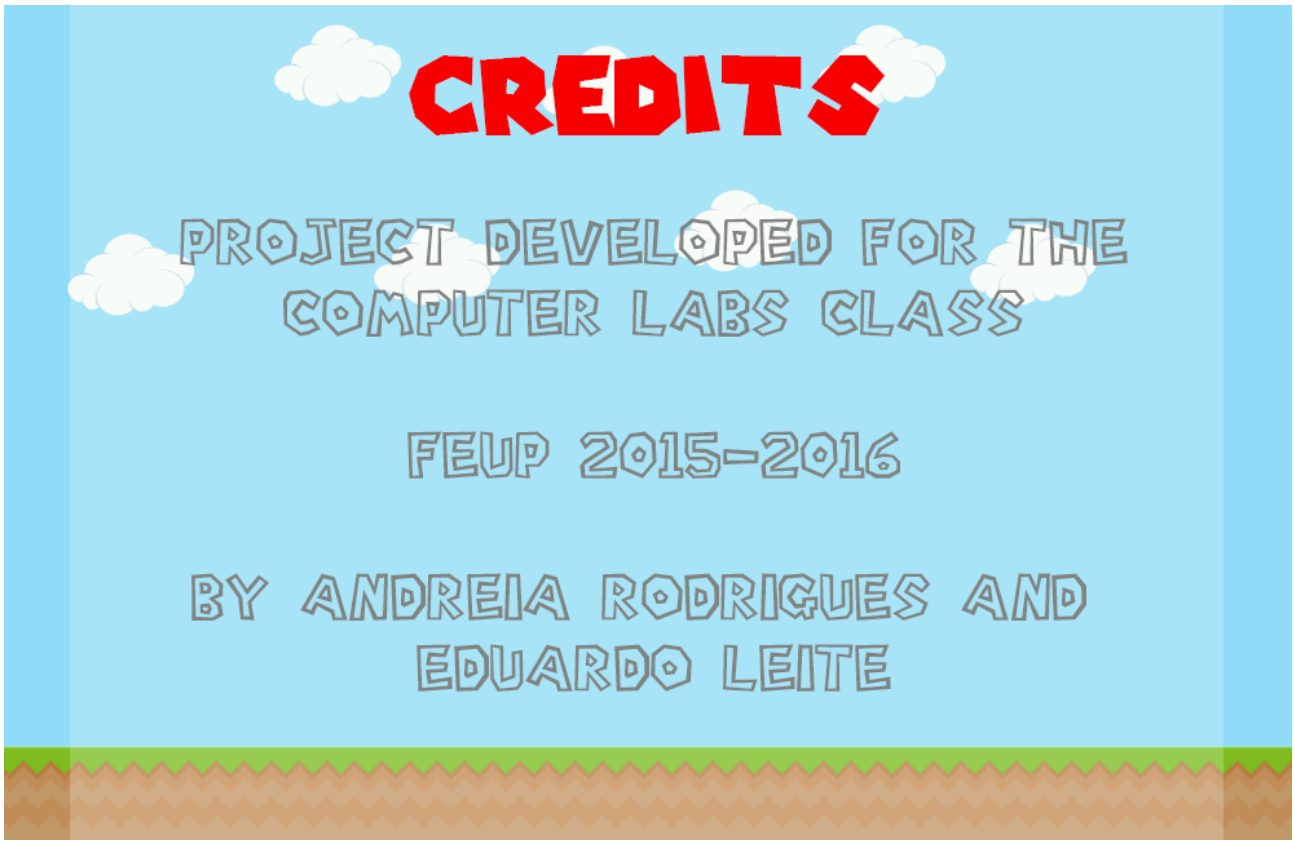


Menu dos controlos



Menu dos High Scores (os valores estão “estranhos” por terem sido mexidos no .txt na fase de

testes)



Menu dos créditos

Estrutura e organização do código

- Bitmap (8%)

Contêm todo o código referente ao desenho e manipulação de bitmaps, permite carregar uma imagem bmp, desenha-la no ecrã, filtrar uma determinada cor a uma imagem e rodar uma determinada imagem. O nosso trabalho foi baseado no trabalho desenvolvido pelo Henrique Ferrolho, com a adição da função que dá flip a um bitmap.
Realizado em conjunto 50:50.

- Boxes (6%)

Contém toda a lógica necessária ao desenho, criação e verificação de colisões para caixas e caixas destrutíveis.
Realizado por Eduardo Leite.

- Camara (4%)

Onde está implementada a função que simula uma câmara, permitindo que a personagem se mantenha no mesmo lugar, movendo todos os objetos existentes à sua volta e criados ao longo do nível.
Realizado por Eduardo Leite.

- Character (10%)

Contém toda a lógica necessária ao desenho e criação da personagem, assim como toda a lógica referente à física, update das suas propriedades, tratamento de input e animações da personagem.
Realizado por Eduardo Leite.

- Clouds (1%)

Contém toda a lógica necessária ao desenho e criação de nuvens.
Realizado por Andreia Rodrigues

- Coins (1%)

Contém toda a lógica necessária ao desenho, criação e colisão da personagem com as moedas.
Realizado por Eduardo Leite.

- Enemy (8%)

Contém toda a lógica necessária ao desenho, criação e colisão com os inimigos, assim como a lógica que os move e das animações.
Realizado por Eduardo Leite.

- Flag (1%)

Contém toda a lógica necessária ao desenho e criação da bandeira final.
Realizado por Andreia Rodrigues.

- GameLoop (20%)

Contém o ciclo onde está a deteção de todas as interrupções dos periféricos, assim como o controlo dos estados e estruturas do jogo. É na função start game que é feita toda a gestão de inputs do utilizador e as mudanças de estado do jogo como resultado destes. É aqui que é feito load de todos os bitmaps necessários e impressão destes no ecrã, tal como as verificações de colisões da personagem com os objetos á sua volta, etc.
Realizado em conjunto 50:50.

- Graphics (5%)

Onde está armazenado o double buffer e todas as funções que permitem manipula-lo, assim como todos os outros elementos relacionados com a placa gráfica, que permitem inicializa-la, aceder á memória de vídeo e ao segundo buffer, aceder ás propriedades do modo de vídeo com que é inicializada, e preencher todo o ecrã com uma cor especifica.
Realizado por Andreia Rodrigues, as funções RGB e Fill Display foram utilizadas do trabalho realizado pelo Henrique Ferrolho.

- Keyboard (8%)

Onde são definidas as funções que vão permitir interrupções do teclado e fazer o seu devido tratamento. As interrupções vão alterar os estados do jogo e o movimento da personagem.
Realizado em conjunto 50:50.

- LoadGame (3%)

Onde são carregados todos os objetos do nível que têm posições dinâmicas, sempre que necessário iniciar o nível.
Realizado por Andreia Rodrigues.

- Mouse (10%)

Onde são definidas as funções que vão permitir interrupções do rato e as que fazem o tratamento destas. Estas funções vão reconhecer e tratar a posição e estado do rato, para que seja possível interagir com o menu e com as caixas destrutíveis em modo de jogo.
Realizado por Andreia Rodrigues.

- Platform (6%)

Contém toda a lógica necessária ao desenho, criação e verificação de colisões para plataformas.
Realizado por Eduardo Leite.

- project (1%)

Onde se inicializa o programa.

- RTC (2%)

Contém toda a lógica necessária à leitura e tratamento da data no RTC, utilizada para os High Scores. A função que converte de BCD para INT foi retirada de um post do Stack Overflow. "http://stackoverflow.com/questions/28133020/how-to-convert-bcd-to-decimal"

Realizado por Andreia Rodrigues.

- ScoreManager (2%)

Onde está implementada a gestão de scores e a sua leitura e escrita para ficheiros.
Realizado por Eduardo Leite.

- Spikes (2%)

Contém toda a lógica necessária ao desenho, criação e verificação de colisões para picos.
Realizado por Eduardo Leite.

- Timer (3%)

Onde são definidas as funções que vão permitir interrupções do timer e as que fazem o tratamento destas. Realizado em conjunto 50:50.

- VBE (1%)

Onde é inicializado o modo gráfico
Realizado por Andreia Rodrigues.

Detalhes de Implementação

LOAD

O nosso projeto é completamento controlado pelo código escrito no GameLoop.c. Começa pela existência de uma máquina de estados que controla o Screen que está ativo a cada momento.

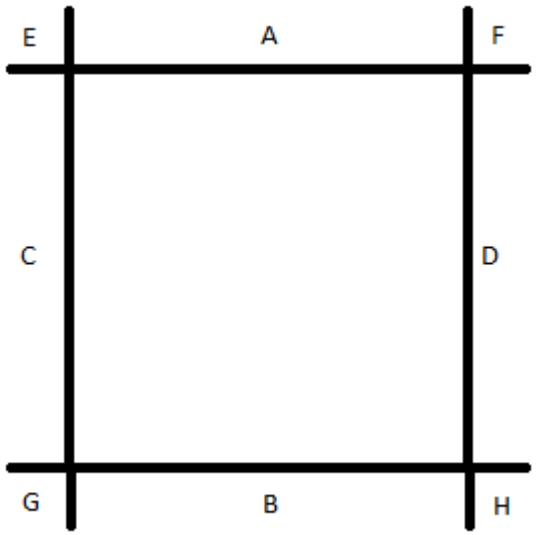
Depois são carregadas todas as imagens e inicializadas todas as variáveis necessárias ao controlo do programa.

A partir daí o jogo entra num ciclo de Update Draw do qual só sairá após dada uma instrução para tal. Este sistema é gerido através de um sistema de deteção de eventos e das interrupções do clock.

UPDATE

A parte do update no Main Menu faz apenas update do rato para verificar a interação com o mesmo e verifica se a tecla ESC foi premida para terminar a execução do programa. Nos restantes Screens apenas vão sendo processadas as variáveis de desenho da interface e também se verifica o uso da tecla ESC para voltar ao menu principal.

No modo de jogo o código de update começa por atualizar a informação relativa à personagem e daí parte para a verificação de colisões com os diferentes objetos, sendo que a posição do esqueleto é atualizada imediatamente antes desta.

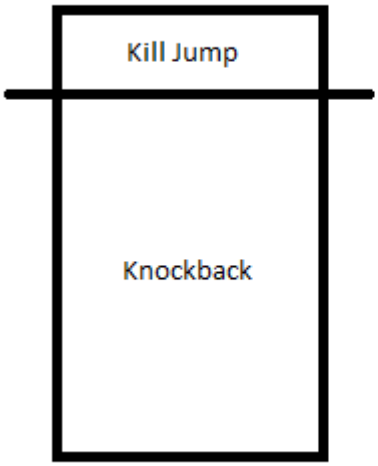


O programa utiliza as coordenadas relativas entre a personagem e os objetos e a velocidade da personagem para identificar e tratar 8 diferentes tipos de colisões diferentes com a maior velocidade possível, tornando a execução do programa mais rápida e eficientes. A implementação deste método pode ser encontrada em boxes.c em platform.c

As colisões foram bastante pensadas e trabalhadas, não são completamente perfeitas, mas cobrem diversos casos de utilização.

Nos casos A B C e D em que a personagem está apenas em contato com uma superfície do objeto, o alinhamento é feito utilizando a posição relativa entre os dois e a velocidade nesse eixo, sendo este alinhamento vertical nos casos A e B e horizontal nos casos C e D.

Os casos E F G e H foram mais complicados, começamos por verificar através da velocidade vertical se o alinhamento teria de ser necessariamente horizontal, por exemplo se se der uma colisão do tipo E ou F enquanto a personagem sobe, então o alinhamento terá de ser feito na horizontal pois este não pode estar a atingir o objeto por cima. Caso isto não aconteça, é feita uma comparação entre a distância horizontal e vertical ao eixo em questão para decidir o tipo de alinhamento.



O programa diferencia duas colisões entre a personagem e os esqueletos.

Existe o caso em que a colisão acontece entre a personagem e o topo do esqueleto, em que o esqueleto é morto.

E o caso em que a personagem perde Health Points e é empurrada para trás

No caso de haver uma colisão com um inimigo ou com os picos, sendo que o alinhamento é substituído por um KnockBack, a implementação torna-se um pouco mais fácil. É necessário distinguir os dois casos acima referidos através da posição relativa entre o inimigo e a personagem (no caso do esqueleto) e depois aplicar o knockback de acordo com a velocidade horizontal da personagem no momento do impacto.

Foi também criada uma variável auxiliar "horizontal_effect" para distinguir o movimento vindo do Input, do movimento vindo destas colisões, sendo que é necessário retirar o controlo da personagem, até esta atingir o chão ou uma plataforma, durante a duração deste movimento, para que ele não seja substituído pelo movimento natural da personagem.

No final do ciclo de update é feita a verificação das três condições de terminação, o tempo decorrido, os pontos de vida do jogador e a conclusão do nível chegando à bandeira.

DRAW

Depois de realizado o update procede-se ao desenho dos elementos, este desenho é realizado por layers para que os elementos não se sobreponham de forma indesejada. O desenho

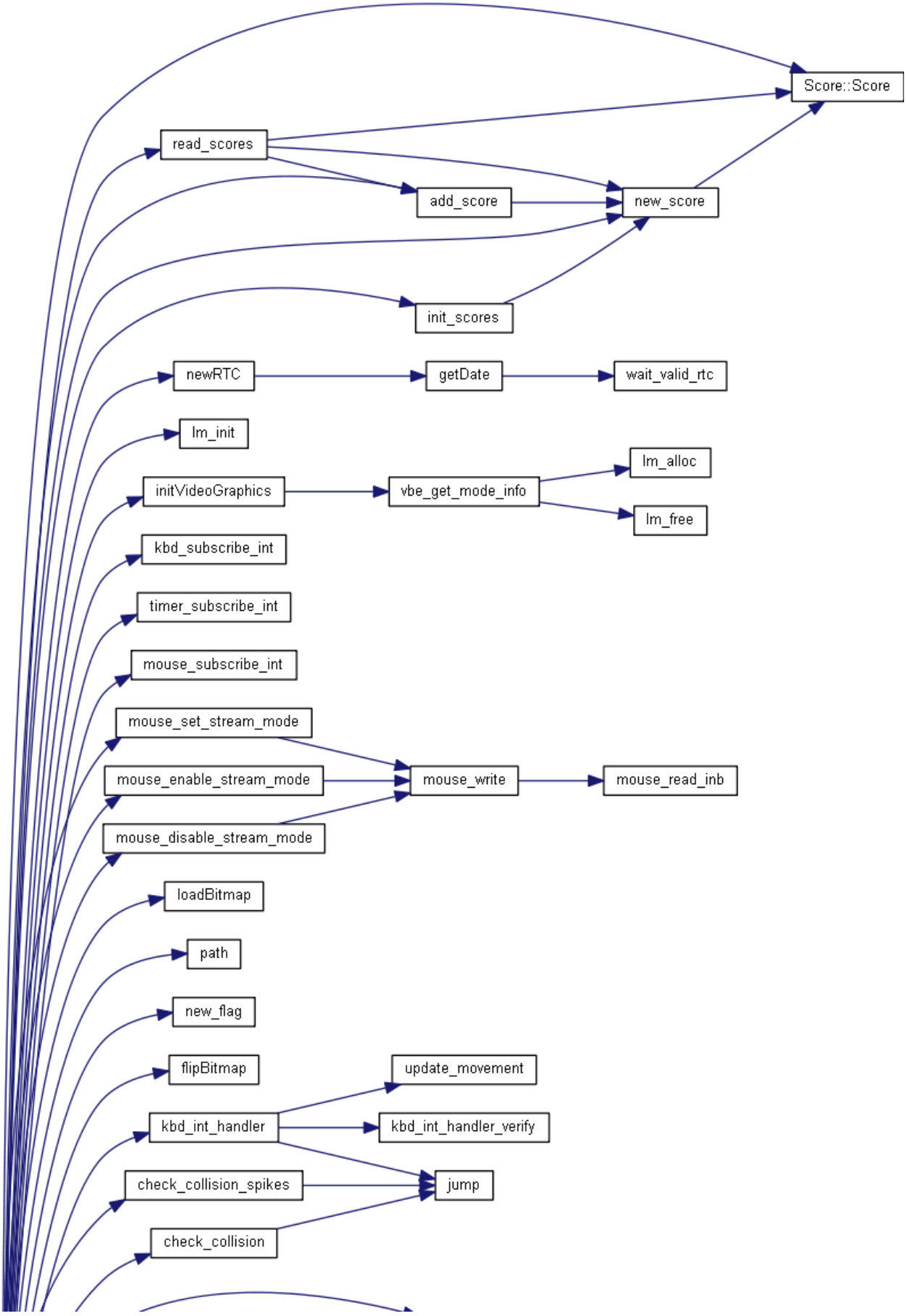
do rato é realizado em último lugar e utiliza uma função específica para filtrar a sua cor de fundo.

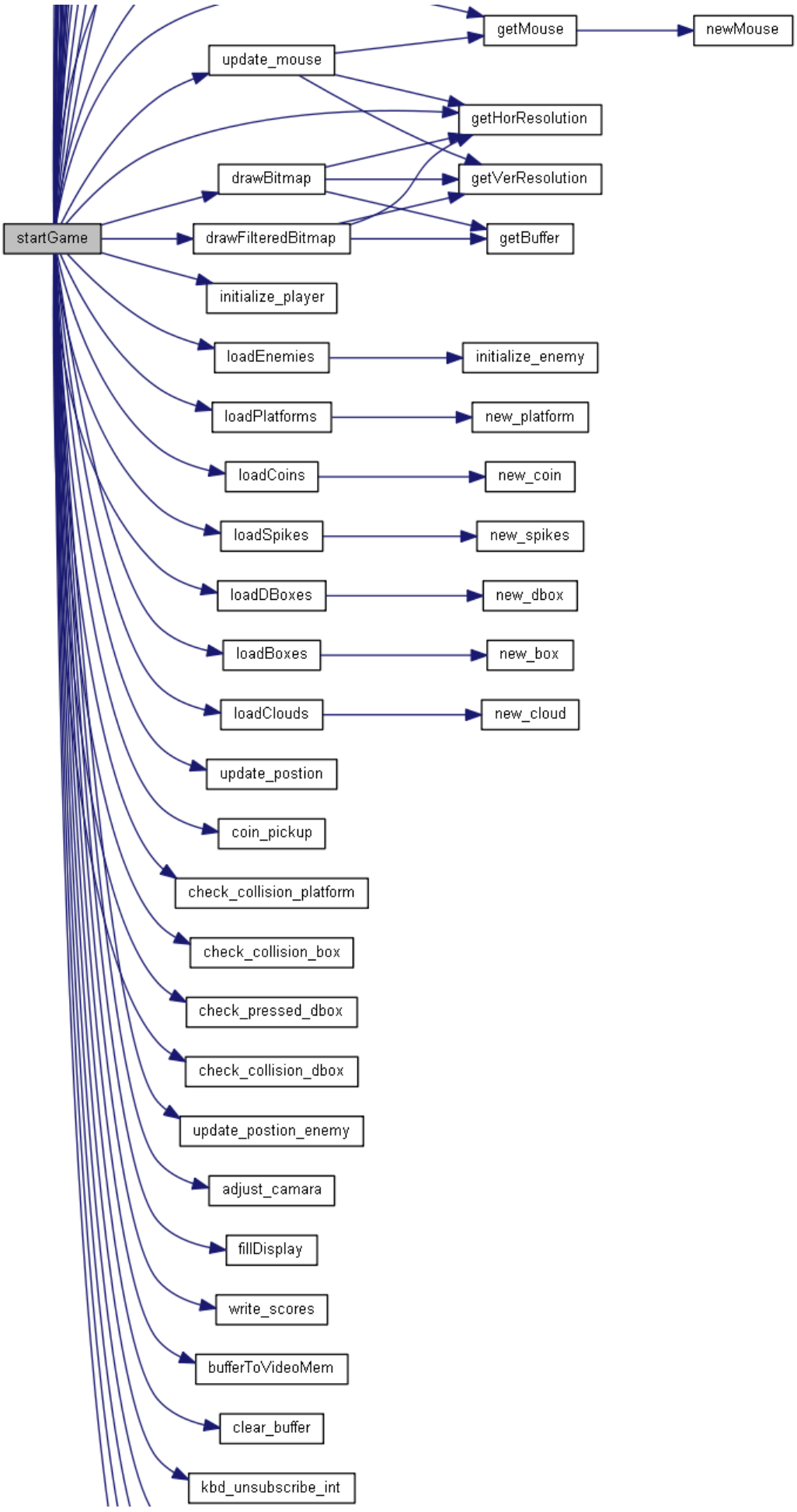
O programa utiliza também um sistema de fonts para desenhar todas as variáveis necessárias, o tempo decorrido, o número de moedas coletadas e a informação sobre os scores.

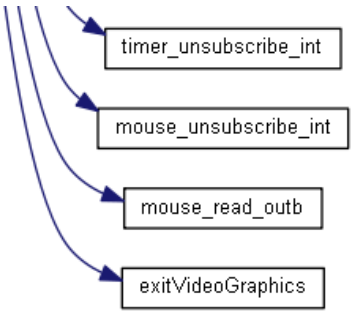
UNLOAD

No final do programa é feito o unsubscribe das interrupções e a saída do modo gráfico.

Call Graph


Score::Score
read_scores
add_score
new_score
init_scores
newRTC
getDate
wait_valid_rtc
lm_init
lm_alloc
initVideoGraphics
vbe_get_mode_info
lm_free
kbd_subscribe_int
timer_subscribe_int
mouse_subscribe_int
mouse_set_stream_mode
mouse_enable_stream_mode
mouse_write
mouse_read_inb
mouse_disable_stream_mode
loadBitmap
path
new_flag
flipBitmap
update_movement
kbd_int_handler
kbd_int_handler_verify
check_collision_spikes
jump
check_collision


getMouse
newMouse
update_mouse
getHorResolution
drawBitmap
getVerResolution
startGame
drawFilteredBitmap
getBuffer
initialize_player
loadEnemies
initialize_enemy
loadPlatforms
new_platform
loadCoins
new_coin
loadSpikes
new_spikes
loadDBoxes
new_dbox
loadBoxes
new_box
loadClouds
new_cloud
update_postion
coin_pickup
check_collision_platform
check_collision_box
check_pressed_dbox
check_collision_dbox
update_postion_enemy
adjust_camara
fillDisplay
write_scores
bufferToVideoMem
clear_buffer
kbd_unsubscribe_int


timer_unsubscribe_int
mouse_unsubscribe_int
mouse_read_outb
exitVideoGraphics

Conclusão

Fazer este trabalho ajudou imenso a consolidar o que aprendemos nas aulas, por um lado algumas coisas foram aperfeiçoadas e por outro alguns conceitos que não tinham ficado completamente esclarecidos, como por exemplo o double buffering, e que ficaram agora completamente entendidos.

Estamos satisfeitos com o nosso trabalho, sentimos que mostramos que somos capazes de realizar um projeto utilizando o que aprendemos nas aulas, complementado com alguma pesquisa da nossa parte, com destaque para as colisões. Temos apenas pena de não ter tido tempo para implementar o UART, pois sem dúvida parecia interessante.

Contribuição para o relatório

O relatório foi feito em conjunto, sendo que ambos contribuimos da mesma forma para a realização do mesmo.

Avaliação da disciplina.

Gostamos imenso da maneira como tanto o professor das teóricas e das práticas agiram, a disciplina foi interessante e desafiadora. Contudo sentimos que os guiões poderiam ser mais específicos e com mais exemplos de código, especialmente no que toca ao assembly e à placa gráfica, para que as aulas práticas não fossem tanto uma competição pela atenção do professor.

Também sentimos que 6 créditos eram pouco para o tempo requerido por LCOM face ao tempo necessário às restantes disciplinas. Por último, achamos injusto o facto das turmas de Sexta-Feira terem uma semana completa desde a aula teórica para realizar os labs, enquanto as de Segunda-Feira terem apenas 3 dias.