UNIVERSIDADE FEDERAL DE MINAS GERAIS

CIÊNCIA DA COMPUTAÇÃO – 2018/2

Disciplina: AEDS-1

Professor: Pedro O. S. de Melo

Aluno: Daniel Souza de Campos

Introdução

Em uma galáxia não muito distante, esse trabalho tem o objetivo de explicar o funcionamento do programa desenvolvido pelo aluno Daniel Souza de Campos assim requisitado pelo professor Pedro Olmo como Trabalho Prático da disciplina de AEDS-1. O trabalho consiste em programar uma versão do jogo **ENDURO** utilizando a biblioteca **ALLEGRO** e os conhecimentos adquiridos ao longo do semestre.

Como funciona

1. Variáveis globais e constantes

O programa possui variáveis definidas entre as linhas 12 e 18. São elas:

- NUMCARS: Número de carros utilizados como oponentes.
- INC: Quantidade de pixels que o carro do jogador irá andar para os lados quando necessário.
- INCACELERA: Incremento na velocidade dos carros oponentes quando o jogador quiser ir mais rapidamente.
- INCFREIA: Incremento negativo na velocidade dos carros oponentes quando o jogador decidir ir mais devagar.
- INC2: Velocidade base dos carros oponentes.
- INCLADO: Incremento para o lado dos carros oponentes à medida que descem a tela dependendo de sua posição inicial.
- TAXA: Número mínimo necessário a ser alcançado pelo contador que define a cada quanto tempo um ou mais carros serão gerados na tela.

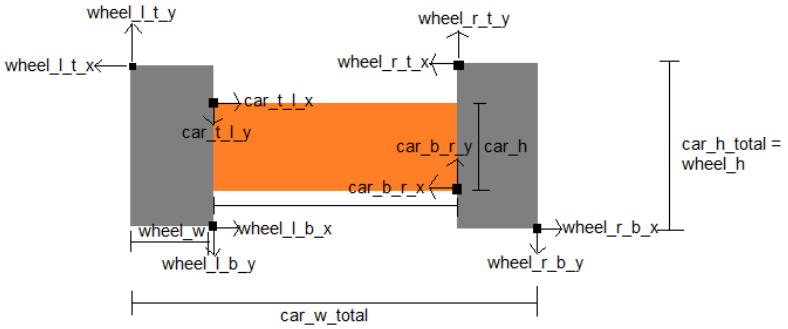
Constantes:

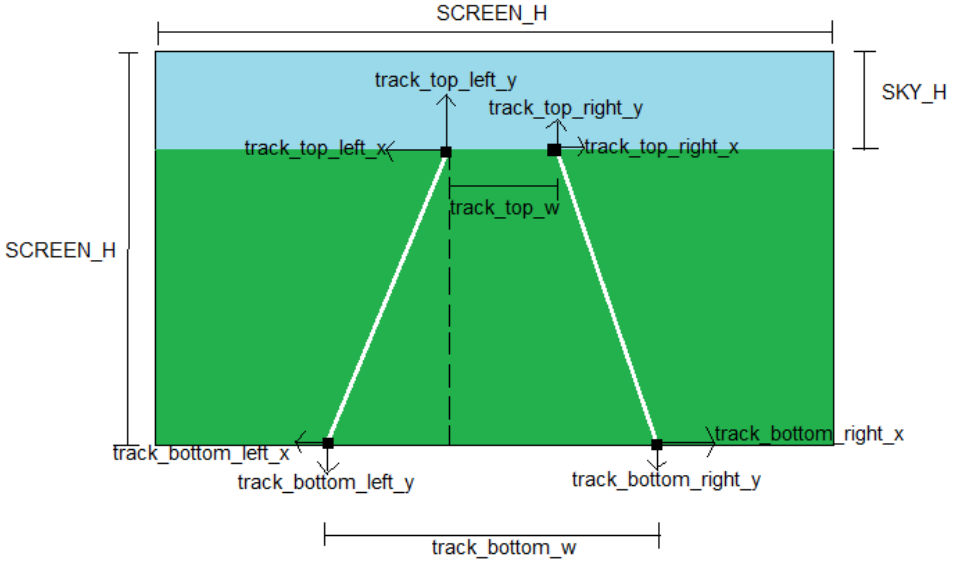
- FPS: Quantidade de quadros por segundo em que o timer do jogo se baseará.
- SCREEN_W: Comprimento total da tela.
- SCREEN_H: Altura total da tela.

Da linha 24 até 46 são variáveis globais necessárias tanto para ajudar a desenhar os traçados da pista quanto a desenhar o carro. Destaque para *ymin*, *THAN_THETA*, usados para calcular as proporções dos carros, *size_32* e *score* que são usadas para mostrar as mensagens na tela.

A seguir, na linha 48, começa a definição de uma *struct* de nome **Carro** que contém todas as variáveis necessárias para definir os tamanhos e posições de todos os carros que usarão essa *struct* como base.

- Car_w_total: comprimento máximo do carro.
- Car_h_total: altura máxima do carro.
- Car_w: comprimento da carroceria do carro.
- Car_h: altura da carroceria do carro.
- Car_t_l_x: posição x do topo esquerdo do carro (resumo de car_top_left_x);
- Car_t_l_y: posição y do topo esquerdo do carro.
- Car_b_r_x: posição x da base direita do carro.
- Car_b_r_y: posição y da base direita do carro.
- Wheel_w: Comprimento total da roda.
- Wheel_h: Altura total da roda que é igual a altura total do carro.
- Wheel_l_t_y: posição y do topo esquerdo da roda esquerda.
- Wheel_l_t_x: posição x do topo esquerdo da roda esquerda.
- Wheel_l_b_y: posição y da base direita da roda esquerda.
- Wheel_l_b_x: posição x da base direita da roda esquerda.
- Wheel_r_t_y: posição y do topo esquerdo da roda direita.
- Wheel_r_t_x: posição x do topo esquerdo da roda direita
- Wheel_r_b_y: posição y da base direita da roda direita.
- Wheel_r_b_x: posição x da base direita da roda direita.
- Xoffset: distância do carro até pista esquerda.
- Y2: altura do triângulo criado pelo carro até o topo da pista.
- X2: comprimento do triângulo formado pela distância entre as linhas da pista dependendo da altura do carro na tela.
- Inclado: incremento lateral para os carros oponentes dependendo da sua posição inicial.
- Cor: cor aleatória para os oponentes e já definida para o jogador.





Na linha 60 e 61 são criadas as variáveis do carro do jogador e o vetor de carros oponentes de tamanho NUMCARS.

2. Funções

A partir da linha 64 começam as funções necessárias para a execução do programa. Em ordem que aparecem:

- *int bounding-box_collision():* Função que confere se o carro do jogador e um carro inimigo colidiram ao comparar as variáveis recebidas por parâmetro, ou seja, se o carro oponente está fora dos limites do carro do jogador e, caso contrário, é considerado uma colisão.

- int gameOver(): Função responsável por mostrar a tela de Game Over quando o usuário colidir o seu carro com um carro inimigo.

  A tela consiste em três mensagens no total. As mensagens “Pontos: n° de pontos” e “Game Over” sempre aparecerão para o jogador. A mensagem “NOVO RECORDE” só aparecerá caso a pontuação obtida pelo jogador for maior que o recorde registrado no jogo. O recorde histórico fica armazenado no arquivo “recorde.txt” e será substituído caso o mesmo seja superado. Isso ocorre entre as linhas 95, onde a variável *arq* está sendo preparada para receber o arquivo, até a linha 107.
  Após isso, o programa fica esperando uma ação específica do usuário que é ou apertar a tecla *ENTER*, *ESCAPE* ou fechar o jogo, sendo que a tecla *ESCAPE* termina o jogo e a tecla *ENTER* reinicia o jogo, tudo isso entre as linhas 111 e 129.

- *void calcDimCarro():* Função responsável por definir todas as variáveis do carro com exceção de *wheel_l_t_x*, *wheel_l_t_y*, *y2*,*x2*,*cor* e *incLado*.

- *float calcX2():* Função responsável por calcular *x2* a partir de *y2* passado por parâmetro. Detalhe que *x2* é calculado pela divisão de *y2* pela variável TAN_THETA para manter as proporções do triângulo já descrito anteriormente.

- *float calcY2():* Função responsável por calcular *y2* que é a diferença entre a altura atual do carro e *ymin* que é igual à SKY_H.

- *void calcTamMaxCar():* Função responsável de definir *car_w_total* como sendo um sexto do *x2* atual do carro e *car_h_total* como sendo um décimo do *y2* atual do carro. Ou seja, o comprimento do carro é igual a um sexto da largura da pista de acordo com a altura em que o carro está e a altura do carro é igual à um décimo da distância do carro até o topo da pista. Essas são as proporções básicas em que os carros seguirão para dar a impressão de aproximação do carro do jogador.

- *void attPosCarro():* Função responsável de incrementar as posições de *wheel_l_t_y* e *wheel_l_t_x* do carro passado por parâmetro de acordo com os incrementos também passados por parâmetros que dependem do movimento do carro, tanto oponente quanto do jogador. Essas variáveis são as que ajudam a definir todas as outras variáveis do carro. É nessa função que ocorre a chamada das funções *calcTamMaxCar()* e *calcDimCarro().*

- *void PosCarro():* Uma das funções mais importantes do programa, responsável por definir as posições iniciais dos carros dos oponentes e

do jogador. Ela recebe o carro em questão por parâmetro e também uma variável inteira *i* que serve para identificar se o carro passado é do jogador (1) ou oponente (2).

Caso seja do jogador, ele define de novo as variáveis *wheel_l_t_y* e *wheel_l_t_x* com valores que deixam o carro centralizado e um pouco acima do fim da tela. Além disso, define a cor do carro como laranja (linhas 195 e 196).

Caso seja oponente, define a variável *wheel_l_t_y* como sendo 1 pixel a mais que o valor de SKY_H. Depois, sorteia um valor inteiro *pos* entre 0 e 4(incluindo o 4) para definir valores pré-estabelecidos de *wheel_l_t_x* simulando 5 faixas de carro na pista. Um *switch* na variável *pos* analisa o seu valor e define *wheel_l_t_x* e o *incLado* do carro como sendo negativo ou positivo de valores diferentes para manter o carro em sua faixa. Feito isso, sorteia uma cor aleatória para esse carro e calcula novos valores de *x2* e *y2* com *calcY2()(*linha 238) e *calcX2()*(linha 239) para o carro em questão e termina de definir suas novas dimensões com *calcTamMaxCar()*(linha 240) e *calcDimCarro()(*linha 241).

- *void init_global_vars():* Função responsável por dar valores às variáveis globais que foram declaradas sem recebem valores diretamente. Além disso, calcula a variável TAN_THETA utilizada no cálculo das dimensões dos carros.

- *void draw_car():* Função responsável por desenhar o carro passado por parâmetro utilizando suas variáveis e que delimitam as rodas e a carroceria do carro em suas respectivas posições e cores utilizando o método *al_draw_filled_rectangle().*

- *void draw_inner_mountain():* Função responsável por desenhar a segunda camada de montanhas com uma cor mais escura de marrom utilizando o método *al_draw_filled_triangle()* também utilizada nas próximas duas funções.

- *void draw_snow_mountain():* Função responsável por desenhar a ponta branca da montanha para dar impressão de neve.

- *void draw_mountains():* Função responsável por desenhar três cadeias de montanhas em três lugares diferentes da tela. Desenha a primeira camada das montanhas com um marrom mais claro e deixa a segunda camada e a ponta branca para as funções *draw_inner_mountain()* e *draw_snow_mountain()* respectivamente.

- *void draw_scenario():* Outra função importantíssima no programa responsável por fazer a desenhar e fazer a chamada de funções responsáveis por desenhar algo na tela. Ela desenha diretamente

o fundo verde representando um gramado (linha 317), o fundo azul claro do céu (linha317), as delimitações da pista (linhas 324-326) e o texto com a pontuação atual do jogador (linhas 331-336). Além de chamar a função *draw_car()* (linha 329).

- *int main():* A principal função do programa como já diz o seu nome. Primeiramente, ela define uma *seed* para o método *rand()* que é utilizada na função *PosCarro()* com o método *srand()* passando o valor retornado pelo método *time().*

  Logo depois, são definidas as variáveis responsáveis por armazenar o *display* da tela, a fila de eventos, o *timer* e a música que será utilizada respectivamente (linhas 343-346) e então faz a chamada do método *init_global_vars()* e *PosCarro()* para definir todas as variáveis e poder desenhar futuramente o carro centralizado e um pouco acima do fim da tela.

  A partir da linha 352 começam as rotinas de inicialização dos *addons* a serem utilizados pelo programa e seus tratamentos de erros como: o próprio *allegro*(*al_init()*), de desenho (*al_init_primitives_addon()*), temporizador (*al_create_timer()*), display (*al_create_display()*), fontes (*al_init_font_addon()*), formatação de fontes (*al_init_ttf_addon()*), funções de áudio (*al_install_audio()*), formatos de arquivos de áudio(*al_init_acodec_addon()*) e canais alocados no *mixer* de áudio (*al_reserve_samples()*).

  Na linha 403 é carregado o arquivo de música definido para ser tocado com o método *al_load_audio_stream()*. Em 409, é adicionado no mixer a própria música e na linha seguinte fica definido que a mesma será tocada em loop com o método *al_set_audio_stream_playmode()* e o parâmetro ALLEGRO_PLAYMODE_LOOP para a respectiva música. Em 412 a música começa a ser tocada com o método *al_set_audio_stream_playing().*

A partir da linha 419 começam a ser definidos os eventos a serem registrados na fila de eventos que será analisada para detectar ou quando o jogador fez um movimento no teclado ou quando o timer estabelecido disparou o seu evento. Na linha 442, o *timer* é iniciado.

Entre 447 e 450 são definidas variáveis que serão utilizadas futuramente no loop infinito que o jogo ficará.

- *int playing*: Enquanto o jogador não decidir finalizar o jogo essa variável permanecerá com o valor 1, caso contrário, será definida como 0.
- *int opCars* e *carsUtili*: Utilizadas para manter salvo o número de carros oponentes utilizados no programa.
- *float count*: Utilizado para ser acrescentado a cada *loop* do *while* seguinte para ser contado até o valor estabelecida por

TAXA para assim poder definir a posição inicial de um ou mais carros inimigos.

- ALLEGRO_EVENT *ev*: Variável responsável por salvar um evento.
- *bool right,left,up,down* e *pause*: Variáveis responsáveis por ajudar a detectar se o usuário apertou e soltou uma tecla. Caso estejam com valores *true*, a variável ainda está sendo apertada e só voltarão a ser *false* quando a tecla em questão for solta.

Na linha 452 é iniciado o bloco da estrutura de repetição *while* que permanecerá em um *loop* enquanto a variável analisada *playing* estiver com o valor igual a 1 representando um valor *true*.

A primeira linha dentro do bloco do *while* é o método *al_wait_for_event()* que espera o disparo de um evento ou pelo teclado ou pelo timer ou pela própria tela. Na linha 456 começa a análise da implementação do pause feita durante a apresentação do programa ao professor.

Se a variável pause estiver com o valor *true* e for detectado que o usuário soltou a tecla P a variável volta a ser false e o jogo despausa. Caso contrário, o programa pula todas as outras linhas do *while* e dá a impressão que o jogo está pausado.

Na linha 464, é analisado se o evento disparado é o aperto de alguma tecla no teclado. Caso verdadeiro, um switch analisa se foi uma das seguintes teclas: SETA PARA CIMA, W, SETA PARA BAIXO, S, SETA PARA ESQUERDA, A, SETA PARA DIREITA, D, ESCAPE ou P. Assim, dependendo da tecla, as variáveis *right, left, up, down* ou pause serão definidas com valores verdadeiros. Caso sejam as teclas seta para esquerda ou A, será analisado se em um possível incremento do carro do jogador para a esquerda o fará progredir a mais do que o delimitado pela linha lateral esquerda da pista. Se falso, a função *attPosCarro()* é chamada passando o carro do jogador como referência e um incremento negativo de INC para que o carro ande para a esquerda.

O mesmo acontece para as teclas seta para a direita e D, mas são analisados casos de incremento para a direita e a linha lateral direita da pista, também chamando a função *attPosCarro()* com incremento horizontal INC positivo para que o carro ande para a direita.

Na linha 507, começa a analise se o evento disparado for quando uma das mesmas teclas analisadas anteriormente for solta e assim *up, down, left* ou *right* recebem o valor *false*.

Já na linha 538, começa a análise se o evento disparado foi o do timer. Caso verdadeiro, começa o incremento de um em um da variável *coun*t que espera ser maior ou igual que TAXA para poder posicionar um carro oponente no topo da tela com *PosCarro()* passando por referência um carro oponente salvo no vetor de Carros oponentes[] e o valor 2 identificando que se refere a um carro oponente.

Logo em seguida, na linha 549, é analisado se a variável *left* permanece com o valor *true* o que identifica que a tecla ainda está sendo segurada pelo usuário e, assim, ele incrementa mais uma vez a posição horizontal do carro do jogador com um valor negativo de INC. Na linha 557, é feito o mesmo mas para a variável right que, caso positivo, incrementa positivamente a posição horizontal do carro do jogador com o valor de INC.

Em 563 é chamado o método *draw_scenario()* para que o cenário fique como fundo e apenas depois disso sejam desenhados os carros por cima.

Entre as linhas 565 e 606 são atualizadas todas as variáveis dos carros oponentes utilizados no momento pelo programa dependendo se o usuário está apertando a tecla *up*, o que aumenta a velocidade com que os carros inimigos se locomovem com a variável INCACELERA, *down*, que diminui a velocidade dos oponentes com a variável INCFREIA, ou se nenhuma das duas estiver sendo apertada é atualizada a posição dos carros inimigos com o valor base de sua velocidade INC2.

Em 580, é verificado se houve alguma colisão entre os oponentes e o carro do jogador utilizando a função *bounding_box_collision().* Caso verdadeiro, é analisado se a função chamada *gameOver()* retorna verdadeiro, o que acontece caso o usuário queira continuar jogando e, assim, pinta o cenário de novo com *draw_scenario().* Caso falso, a variável *playing* recebe 0 o que significa que o programa deverá parar com a quebra do laço do *while*. Independentemente, muitas variáveis são “zeradas” para ou reiniciar o jogo ou para fechá-lo (linhas 588 – 595) e então é redefinido a posição inicial do carro do jogador em 597.

Em 601 é analisado se um carro oponente já ultrapassou os limites da tela na vertical e se está no momento certo de imprimir um carro no topo de acordo com a variável *count* e, caso verdadeiro, a sua posição é redefinida e o score, variável responsável por contar a pontuação do jogador, é incrementada em um. Assim, para cada carro que chegou no final da tela, o score é incrementado e a pontuação do jogador aumenta.

Em 608, caso *count* seja maior que a TAXA, *count* é zerado para poder começar a ser contado de novo.

Ainda no mesmo evento, a tela é atualizada com o método *al_flip_display()* em 612.

Na linha 615, caso o evento seja de fechamento da tela, a variável *playing* também recebe 0.

Assim fecha o *while* onde o programa passará a maior parte do tempo.

A partir da linha 621 são liberadas as alocações de memórias de todas as variáveis do alegro até o fim do programa em 634.