



Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Departamento de Engenharia Informática  
2008/9

Computação Gráfica

**Relatório Final**

**TuxSkater**



**Trabalho elaborado por:**

André Santos - andrels@student.dei.uc.pt- 2006124047  
Nuno Khan - nfkhan@student.dei.uc.pt- 2006131722  
56 Horas

# Introdução

---

O nosso projecto passa por desenvolver um jogo de computador inspirado nos vários jogos de nome “Tux”, obtendo este a denominação de “Tux Skater”.

Este consistirá no famoso “Tux” a passear de Skate por um mundo 3D, mais concretamente um espaço urbano, onde o objectivo do mesmo é apanhar todas as latas de cerveja.

De notar que esta aplicação foi desenvolvida em/para ambiente Linux, mas propriamente em distribuições baseadas em Debian (Ubuntu). Uma explicação de como instalar e iniciar o jogo será apresentada no final do relatório.

# Funcionalidades Implementadas

Aqui está uma tabela especificando algumas das funcionalidades implementadas para a concretização do nosso projecto.

<b>Funcionalidade</b>	<b>Implementada</b>	<b>Condições/instruções para testar</b>	<b>Problemas identificados relativamente a essa funcionalidade</b>
Movimentação do Tux	100% implementada	Carregar nas setas Up/Down para aumentar/diminuir a velocidade do Tux e setas Left/Right para virar	Criar um movimento realista.
Detecção de Colisões	100% implementada	Chocar contra um prédio, relva ou lata de cerveja (neste caso a lata desaparece)	Como detectar o objecto com que colide e qual o lado da colisão
Simulação da passagem do dia para a noite e vice-versa	100% implementada	Deixar o tempo passar	
Mini-mapa (visão ortogonal)	100% implementada	Mostra as posições da cerveja e do Tux	
Iluminação local	100% implementada	Esta de acordo com a simulação da passagem do dia para a noite e vice-versa	
Música de Fundo	100% Implementada	Começa de forma automática e mantém-se em "loop"	Inicialmente usamos a função aplay do linux, mas no final decidimos usar a livreria SDL Mixer
Menu de Introdução	100% implementada	Primeira coisa que aparece ao iniciarmos o jogo, é necessário pressionar uma	Não houve problemas

		tecla para avançar	
Efeito da Lata	100% implementada	De cada vez que apanhamos uma lata temos um efeito	Não houve problemas
Zoom In/Out	100% implementada	Z para zoom OUT e X para zoom IN	
3 Níveis diferentes	100% implementada	Com mapa das posições dos objectos é diferente, e as texturas dos prédios.	Não houve problemas

# Estrutura/Design

---

O “TuxSkater” está dividido 4 ficheiros importantes (e seus respectivos .h).

## **RgbImage.cpp:**

Ficheiro que trata de toda a parte de carregar as texturas para a memória.

## **maps.h:**

Ficheiro que contém os diferentes níveis do Tux.

Um mapa consiste num array em que cada posição identifica um objecto, por defeito os índices são os que se seguem:

- 0 - Estrada
- 1 - Prédio
- 2 - Tux
- 3 - Cerveja
- 4 - Vegetação
- 5 - Espectadores

Outra das predefinições é ter, em cada um dos extremos uma linha/coluna de espectadores e outra de vegetação. Por exemplo:

- 55555
- 54445
- 54245
- 54445
- 55555

Por fim, o número de colunas tem que ser o mesmo que o número de linhas.

## **drawObjects.c:**

Neste ficheiro temos a parte da construção de todos os objectos utilizados na nossa aplicação. É constituído pelos seguintes métodos:

- static void drawBox(GLfloat size, GLenum type):
  - Desenha os prédios e passeios
- void myglutSolidCube(GLdouble size):
  - Função auxiliar da anterior (apenas a chama com os parâmetros necessários)
- void bottle():
  - Desenha uma lata de cerveja
- void surface():

- Desenha uma parte da estrada
- void vegetation():
  - Desenha uma parte da vegetação
- void crowd():
  - Desenha uma parte dos espectadores
- void drawbodyskate(GLdouble size):
  - Desenha o skate
- void drawskate():
  - Função auxiliar da anterior (apenas coloca o objecto com o tamanho desejado e no local pretendido)
- void drawtux():
  - Desenha o nosso Tux. Chama as seguintes funções para a construção de cada um dos pedaços do seu corpo:
    - void draweye1();
    - void draweye2();
    - void drawpeak();
    - void drawhead();
    - void drawarmright();
    - void drawarmleft();
    - void drawlegright();
    - void drawlegleft();
    - void drawbody();

### **tuxSkater.cpp:**

Aqui temos o motor do jogo. Este é constituído pelos seguintes métodos:

- void criaDefineTexturas():
  - Faz o load de todas as texturas necessárias ao jogo. E define os parâmetros das mesmas.
- void defineLuzes():
  - Define a luz ambiente
- void init(void):
  - Aqui é feita a inicialização de alguns parâmetros essenciais à aplicação.
  - É aqui que o método “criaDefineTexturas()” é chamado.
  - É verificado o número de latas no mapa (o score a atingir para passar o mapa).
  - É inicializado o dispositivo de audio.

- void initParticles(int posX, int posZ):
  - Aqui definimos os valores de cada partícula, aleatoriamente, como também a sua posição inicial.
- void \*playme(void \*ss):
  - Este método trata de fazer tocar a música desejada (para o caso da música de fundo, em loop infinito).
- void drawmapa():
  - Neste método é que criamos o mundo e definimos a posição inicial do “Tux” e consequentemente do Observador.
  - Para isso percorremos o array do mapa, e para cada posição verificamos a qual objecto corresponde, consoante o valor guardado nessa posição do array. Inicialmente a variável “start” está a false, isto implica que ainda não foi encontrada a posição inicial do “Tux” (ainda não encontrou uma posição no array com o valor de 2), por isso quando é encontrado o valor 2 no array é calculado os valores de X e Z e a direcção para onde este deve estar virado (onde não existe um prédio ou vegetação imediatamente à frente).
  - De resto é uma questão de verificar qual o valor no array e construir o objecto correspondente.
- void drawScene():
  - Neste método é, primeiramente, feito o desenho do “céu”, que não passa de uma esfera com uma textura do céu aplicada.
  - De seguida é desenhado o Sol e a Lua, é feita a construção do mapa (chama a função “drawmapa”) e eventualmente é desenhado o “Tux”.
- void drawPoints():
  - Este método trata de desenhar, para visualizar no mini-mapa, os pontos correspondentes às posições das cervejas e do “Tux”.
- void drawParticles():
  - Nesta função actualiza-se, para cada partícula, os valores da sua posição e actualiza-se a sua vida, e claro são desenhadas.
- void nevoeiro():
  - Aqui activamos o nevoeiro com os valores de densidade e cor actuais.
- void drawIntro(int type):
  - Aqui, caso esteja actualmente na “Introdução”, “Menu” ou na “Ajuda”, é aplicada a textura correspondente a um plano centrado na origem.
- void renderBitmapString(float x, float y, float z, void \*font, char \*string):

- Este método trata de desenhar na posição desejada, o texto pretendido.
- void display(void):
  - É verificado se o jogo se encontra actualmente na “Introdução”, “Menu” ou na “Ajuda”, e é chamada a função “drawIntro(X)” com o valor de X correspondente.
  - No caso de já se encontrar a jogar, é definido um “Viewport” correspondente ao mini-mapa e chamadas as funções “drawScene()” e “drawPoints()” e mostrado o score actual (número de latas apanhadas).
  - De seguida temos o “Viewport” normal onde se verifica se já apanhou todas as latas, e no caso de ser “true” mostra a mensagem de vitória. Seguidamente é chamada a função “drawParticles()” que actualiza os valores das partículas caso estejam activas.
  - Por fim é chamada a função “drawScene()” e “nevoeiro()”
- int collision(GLfloat posX, GLfloat posZ):
  - Nesta função são verificadas as distâncias do “Tux” para com o objecto em questão, de forma a descobrir com qual das faces colidiu.
  - Retorna o identificador da face
    - 1-TOP
    - 2-RIGHT
    - 3-DOWN
    - 4-LEFT
    - 5-OUT OF BOUNDS
- void defineNewAngles(int type, int countx, int countz):
  - Aqui dependendo do tipo de colisão, se dos lados esquerdo/direito, ou da parte da frente/trás, são calculados os novos ângulos para o observador como para o “Tux”.
  - No caso de este estar fora do mapa, ou preso, dentro de um objecto é verificado se existe um segmento de estrada na área, e actualiza-se as posições do observador e do “Tux”.
- void Timer(int value):
  - Neste método é, primeiramente, verificado se nos encontramos na Introdução, e caso isso se verifique então actualiza-se o timer, até atingir 5 segundos, e o valor de “intro” passa a false.
  - Caso nos encontremos a jogar, então são actualizados os valores do nevoeiro, e a cor da luz ambiente.
  - Depois, caso a velocidade do “Tux” seja maior que 0 e ainda faltem latas para apanhar, verifica-se se está dentro dos limites do mapa. No caso de não estar é chamada a função “defineNewAngles(3,countx,countz)”, onde countx e countz, são os índices correspondentes à posição do “Tux” no array do mapa.



- Caso o “Tux” esteja dentro dos limites do mapa, então é verificado se houve alguma colisão. Em caso afirmativo, verifica-se qual a face com que colidiu, “collision(posX,posZ)”, e chama-se por fim a função que irá definir os novos ângulos do movimento “defineNewAngles()”.
  - No entanto se o objecto com que colidiu é uma lata de cerveja, então são inicializadas as partículas nessa posição, “initParticles(posX,posZ)”, e removida a lata.
- void keyboard(unsigned char key, int x, int y):
  - Aqui, caso esteja actualmente na “Introdução”, “Menu” ou na “Ajuda”, é feita uma espera pela tecla “Enter”. No caso de ter apanhado todas as latas de cerveja, então é feita a passagem para o nível seguinte, feita a reconstrução do nível actual (as posições com valor -1, passam a ter o valor 3), e no caso de já ter percorrido todos os níveis, a variável “menu” passa a true, de forma a mostrar o menu. Para cada passagem de nível é também actualizado o número de latas de cerveja a apanhar, e actualizados os valores das variáveis para os seus valores por defeito.
  - Caso esteja actualmente a jogar, as teclas Z e X ficam activas, para efectuar o zoom in/out.
- void teclasNotAscii(int key, int x, int y):
  - Caso nos encontremos a jogar, então caso seja pressionada a tecla “UP” é aumentado o valor da velocidade e diminuído o valor dos incrementos da rotação do “Tux”. A tecla “UP” só é activa de 0.5 a 0.5 segundos.
  - Depois, caso seja pressionada a tecla “DOWN” é diminuído o valor da velocidade e aumentado o valor dos incrementos da rotação do “Tux”.
  - Caso a velocidade do “Tux” seja menor ou igual que 0 e ainda faltem latas para apanhar, verifica-se se está dentro dos limites do mapa. No caso de não estar é chamada a função “defineNewAngles(3,countx,countz)”, onde countx e countz, são os índices correspondentes à posição do “Tux” no array do mapa.
  - Caso o “Tux” esteja dentro dos limites do mapa, então é verificado se houve alguma colisão. Em caso afirmativo, verifica-se qual a face com que colidiu, “collision(posX,posZ)”, e chama-se por fim a função que irá definir os novos ângulos do movimento “defineNewAngles()”.
  - No entanto se o objecto com que colidiu é uma lata de cerveja, então são inicializadas as partículas nessa posição, “initParticles(posX,posZ)”, e removida a lata.
  - Para o caso de serem pressionadas as teclas “Left” ou “Right”, são actualizados os valores dos ângulos.

# Como instalar e como jogar

---



Em anexo ao código fonte, temos um README.txt onde tem a explicação de como compilar e jogar o TuxSkater.