Lógica de Jogos

Parte 1 – Movendo-se no Cenário

2006 - PUCPR – Tutoria de Jogos – 1º Ano

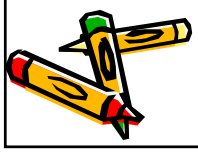
Paulo V. W. Radtke

pvwradtke@gmail.com

http://www.ppgia.pucpr.br/~radtke/jogos

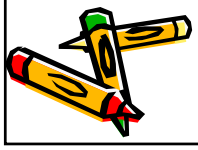
Conteúdo

- Coordenadas de mapa
- Centrando a ação no jogador
- Deslocamento e Gravidade
- Representando um elemento no cenário



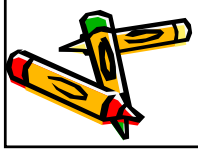
Coordenadas de mapa

- Um jogo trabalha com dois sistemas de coordenadas:
  - Coordenadas da tela: físico
  - Coordenadas de mapa: lógico



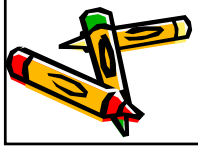
Coordenadas de mapa

- As coordenadas da tela são ditas físicas por representarem pontos mostrados na tela.
- Possuem valores fixo, variando de 0 a um valor máximo (ex: 639).



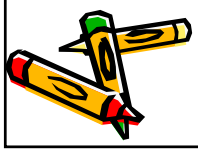
Coordenadas de mapa

- Já as coordenadas do mapa são ditas lógicas por representarem o mundo do jogo.
- Isto ocorre devido ao mapa do jogo ser maior do que é possível de ser mostrado na tela.

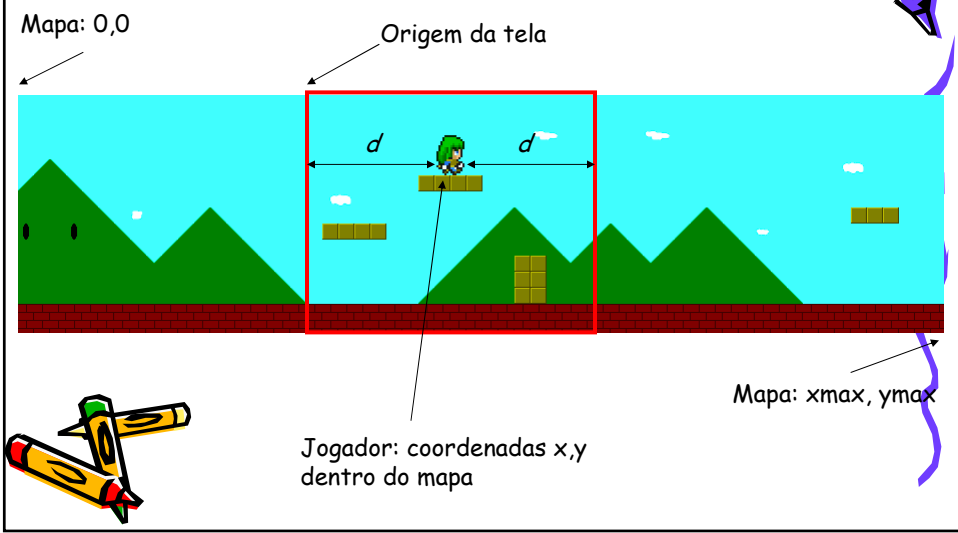


Coordenadas de mapa

- Assim, a ação do jogo passa-se nas coordenadas do mapa.
- O que o jogo faz é mostrar na tela a projeção de parte do mapa.
- Normalmente, esta projeção é centrada no jogador.



Projeção do Mapa na Tela - Caso Geral

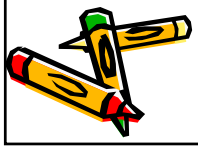


Projeção do Mapa na Tela - Caso Geral

- Neste caso, o jogador fica centralizado na tela.
- Tanto o jogador como inimigos possuem uma posição **x,y** dentro do mapa.
- Com a posição x,y do jogador e as dimensões do mapa, calculamos a origem da tela em coordenadas do mapa.

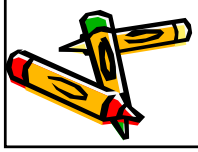
Projeção do Mapa na Tela - Caso Geral

- Desta forma, precisamos conhecer as seguintes variáveis:
  - **x,y**: coordenadas do jogador no mapa
  - **xmax**, **ymax**: dimensões do mapa.
  - **xtela**, **ytela**: dimensões da tela.
  - **ljog**, **ajog**: a largura e altura do jogador em pontos.



Projeção do Mapa na Tela - Caso Geral

- Conhecendo estes valores, podemos calcular **xorig** e **yorig**, a origem da tela no mapa.
- Tais valores são fixos durante o jogo e conhecidos pelo programador.

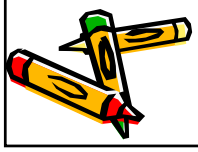


Projeção do Mapa na Tela - Caso Geral

- Assim, temos que:

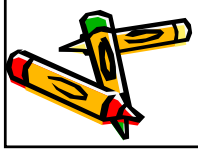
$$xorig = xjog + \frac{ljog}{2} - \frac{xtela}{2} = xjog + \frac{ljog - xtela}{2}$$

$$yorig = yjog + \frac{ajog}{2} - \frac{ytela}{2} = yjog + \frac{ajog - ytela}{2}$$



Projeção do Mapa na Tela - Caso Geral

- O mapa então é desenhado na tela com a origem em **xorig** e **yorig**.
- Com a Chien2DMapy, basta passar estes valores como coordenadas de mapa.



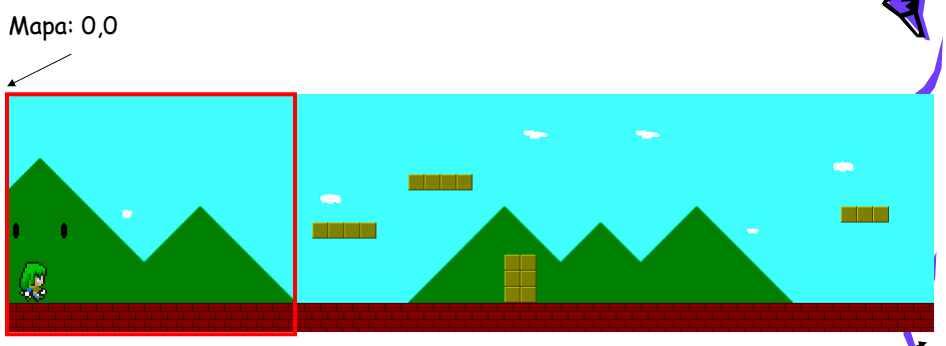
Projeção do Mapa na Tela - Caso Geral

- Para desenhar os elementos na tela (jogador, inimigos, etc), basta subtrair o valor de **xorig** e **yorig** das coordenadas **x**,**y** do elemento.

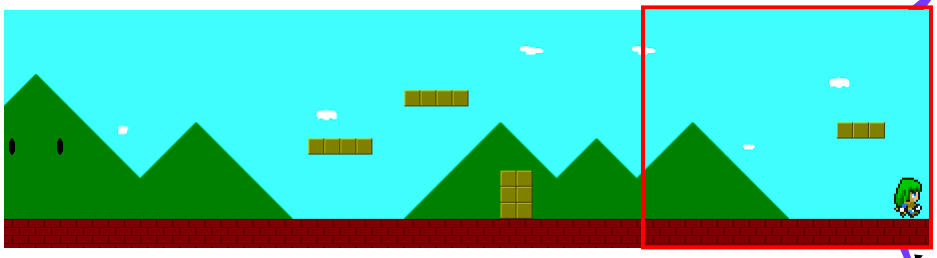
$$x' = x - xorig$$

$$y' = y - yorig$$

Projeção do Mapa na Tela - Caso Especial



Projeção do Mapa na Tela - Caso Especial

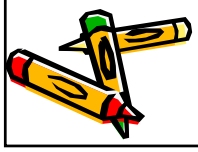


Projeção do Mapa na Tela - Caso Especial

- Os casos especias ocorrem nos cantos do mapa.
- Nestes casos, xorig e yorig são calculados de maneira diferente para que não se tente desenhar fora do mapa.

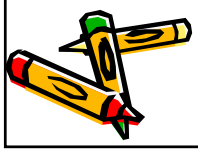
Projeção do Mapa na Tela - Caso Especial

- As condições para que estejamos em um caso especial devem ser verificadas INDEPENDENTEMENTE para cada eixo.
- Assim, é possível que no eixo **x** estejamos em um caso especial, mas não no eixo **y**.



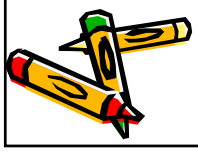
Projeção do Mapa na Tela - Caso Especial

- Para o eixo x:
  - Se o **xorig** calculado for menor que 0, então **xorig** deve ser associado a 0.
  - Se por outro lado o **xorig** for maior que **xmapa**, então ***xorig=xmapa-xtela***.
  - Caso contrário, mantém-se o **xorig** calculado.



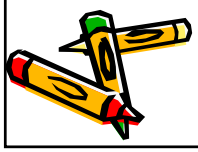
Projeção do Mapa na Tela - Caso Especial

- Para o eixo y fazemos o mesmo:
  - Se o y**orig** calculado for menor que 0, então y**orig** deve ser associado a 0.
  - Se por outro lado o y**orig** for maior que y**mapa**, então y***orig=ymapa-ytela***.
  - Caso contrário, mantém-se o y**orig** calculado.



Projeção do Mapa na Tela - Caso Especial

- Obviamente, os elementos (jogador, inimigos, etc) devem, obrigatoriamente, verificar se eles saem do mapa ao se deslocar.



Deslocando-se no Mapa

- Para um elemento deslocar-se no mapa, este precisa conhecer algumas informações:
  - As suas dimensões.
  - A sua coordenada de referência dentro do sprite original (caso este não cubra o sprite inteiro).

Deslocando-se no Mapa

- Como exemplo, vamos fazer este personagem deslocar-se no cenário:

Deslocando-se no Mapa

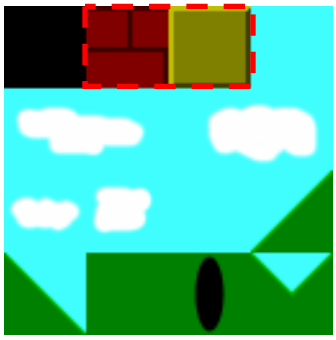
- Ao deslocar o elemento no mapa, testamos se o mesmo não colide com o cenário, baseado no seu **bounding-box**.
- O **bounding-box** é o retângulo que envolve o elemento, que neste caso, tem 28x40 pixels.

Deslocando-se no Mapa

- O tileset utilizado para desenhar o mapa era o seguinte:

Deslocando-se no Mapa

- Porém, apenas dois blocos são "ativos", nos quais o jogador pode andar ou colidir:



Deslocando-se no Mapa

- Estes blocos possuem os código 1 e 2.
- Para deslocar o jogador, basta testar se o seu **bounding box** colide com o **bounding box** de um destes blocos no cenário.
- Para recuperar os blocos para um vetor de blocos, temos o método **copiaCamadaMapa** da Chien2DMappy.

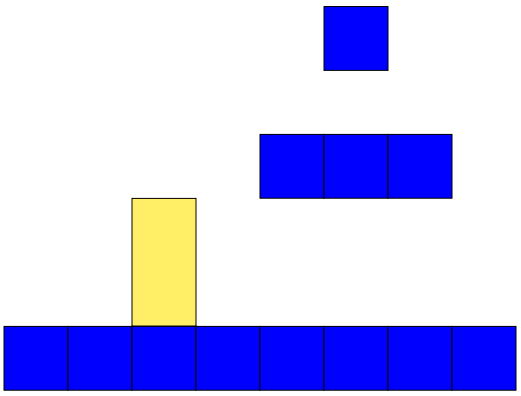
Deslocando-se no Mapa

- Para representar este personagem, precisamos de dois sprites 32x32.
- Assim, o ponto de referência será um pixel dentro dos sprites.

Deslocando-se no Mapa

- Tendo o mapa podemos calcular a posição dos blocos e verificarmos a intersecção do bounding box dos blocos cobertos pelo elemento com o elemento em si.

Deslocando-se no Mapa

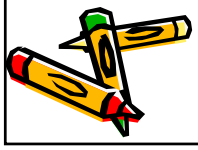


Deslocando-se no Mapa

- A maneira mais simples de realizar o movimento é testar se o jogador pode, ou não, se posicionar na próxima posição calculada.
- Se puder, o jogador tem sua posição mudada, dando a impressão de movimento.

Gravidade

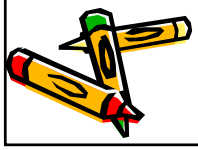
- Em alguns casos, como em jogos *side-view*, faz-se necessário a simulação de gravidade.
- Assim, quando um jogador pula (ganha velocidade), ele é trazido para baixo automaticamente pelo jogo.
- O mesmo ocorre quando ele cai em buracos.



Gravidade

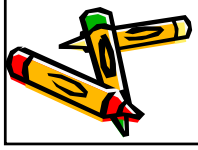
- O processo segue a fórmula clássica de física para o cálculo da velocidade, aonde ***a*** é a aceleração e ***t*** é o tempo passado:

$$v = at^2$$



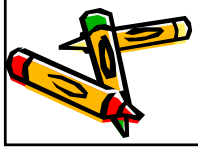
Gravidade

- Esta equação calcula a aceleração de um corpo em queda livre.
- Ela serve tanto para calcular a velocidade da queda como para reduzir a velocidade de um elemento ao dar-se o impulso para um pulo.

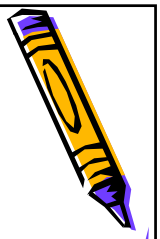


Gravidade

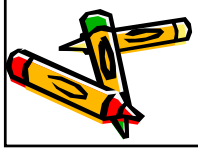
- Assim, precisamos de duas informações para aplicarmos a gravidade em um elemento:
  - A velocidade inicial dele.
  - O tempo passado na ação.
- Para simularmos o atrito do ar, limitamos a velocidade máxima do elemento.



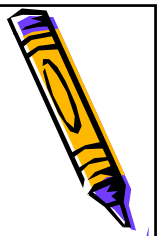
Representando um Elemento



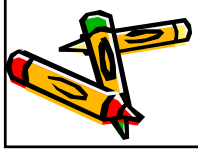
- Para representarmos um elemento, utilizaremos uma estrutura de dados com diversas informações.
- Estas informações são usadas tanto para calcular o deslocamento do personagem como para simular a gravidade.



Representando um Elemento



- Dados do elemento:
  - **x**,**y**: posição do ponto de referência do elemento no mapa.
  - **lele**, **aele**: largura e altura do elemento em pixels.
  - **tempo**: tempo passado desde o início da queda.
  - **vini**: velocidade inicial da queda/pulo.



Próxima Aula

- Na próxima aula, veremos um exemplo incremental que:
  1. Posiciona um elemento no cenário.
  2. Testa colisão.
  3. Simula a gravidade.
  4. Encera a partida ao chegar no final do cenário.
  5. Controla a animação.