Exercício 3 – Objetos em Movimento, Filas e Rotação

Retire as Faces e a função State::AddObject do seu trabalho. A partir daqui, introduziremos entidades definitivas do nosso jogo.

1. Alien: Seguindo os Cliques

Alien (herda de GameObject)
+ Alien (x : float, y : float, nMinions : int) + ~Alien () + Update (dt : float) : void + Render () : void + IsDead () : bool
- Action : class (ver abaixo) - sp : Sprite - speed : Point - hp : int - taskQueue : std::queue<Action> - minionArray : std::vector<Minion>

Action (classe privada em Alien)
+ Action (type : ActionType, x : float, y : float)
+ ActionType : enum (constantes MOVE e SHOOT) + type : ActionType + pos : Point

Alien é um objeto controlado pelo mouse. Ele executa uma sequência de ações, uma a uma, contidas numa fila. Quando um dos botões do mouse

é pressionado em determinado ponto da tela, o Alien cria uma ação e a enfileira. Além disso, ele mantêm um array de objetos filhos (Minions), os quais ele deve atualizar e renderizar.

Uma Action, por sua vez, é dada por um tipo (mover ou atirar), e uma posição para onde o Alien deve se deslocar, ou onde ele deve atirar. O segundo caso, o Alien atirar, ficará pendente Essa classe só é visível para Alien.

```
> Alien (x : float, y : float, nMinions : int)
```

Carrega o Sprite, seta box e inicializa as outras variáveis. Em seguida, devemos popular o array de Minions com alguns destes objetos, espaçados igualmente. Enquanto não tiver certeza que o Alien funciona como desejado, não popule o array.

```
> Update(dt : float) : void
```

Há duas etapas para o comportamento de Alien: Primeiro, checamos se houve input que gere uma ação: clique do botão esquerdo do mouse para um tiro, ou direito para movimento. Se sim, enfileiramos uma ação com a posição do clique - lembre-se do ajuste da câmera.

Feito isso, devemos executar ações pendentes. Checamos se há pelo menos uma ação na fila. Se houver, checamos o tipo. Para ações de movimento, devemos calcular velocidades nos eixos x e y de forma que o Alien se mova em linha reta até aquele ponto, e que o módulo da velocidade dele seja sempre constante.

Isso tudo é caso o Alien não vá chegar ao destino no próximo frame. Se ele estiver a essa distância mínima da posição, devemos colocá-lo lá imediatamente e dar a ação como realizada (tirar da fila). Há dois motivos para isso. Primeiro, se deixarmos o Alien andar na velocidade total, ele vai passar do ponto, e tentar voltar no frame seguinte, no qual ele vai passar de novo.

Ele provavelmente não vai se estabilizar, o que nos leva ao segundo motivo: Trabalhando com floats, nunca usamos '==’. É muito improvável que dois floats que calculamos sejam iguais. Em vez disso, usamos '>’ e/ou '<’ para estabelecer ranges aceitos. Novamente, o movimento se encerra se a distância for menor do que o que o Alien vai mover no frame.

Caso a ação seja de tiro… por enquanto, apenas tire a ação da fila. Precisamos implementar mais algumas coisas antes.

`> Render() : void`

Deve renderizar a si mesmo e chamar a renderização de cada Minion.

`> IsDead() : bool`

O Alien morre quando seu HP chega a zero.

No construtor de State, instancie um Alien em 512,300 e coloque-o no vetor de objetos. Teste se ele se move corretamente, independente da posição da câmera. Se sim, vamos ao...

2. Minion: Objeto em Órbita

Minion (herda de GameObject)
+ Minion (minionCenter : GameObject*, arcOffset : float = 0) + Update (dt : float) : void + Render () : void + IsDead () : bool + Shoot (pos : Point) : void
- center : GameObject* - sp : Sprite - arc : float

Nem todo objeto do nosso jogo pode ter seu movimento controlado: Alguns são guiados por AI e tomam decisões em um contexto, e outros seguem regras simples indefinidamente, definidas no seu Update. Minion é o segundo caso, orbitando um GameObject dado até ser deletado.

Além do seu Sprite, Minion carrega um ponteiro para um objeto e um ângulo, arc, que indica o arco da circunferência (a órbita) já percorrido. Usando esses dois, podemos atualizar a box do Minion.

`> Minion (minionCenter : GameObject*, arcOffset : float = 0)`

Os argumentos do construtor são o objeto que devemos orbitar e o ângulo inicial. Carregamos o Sprite e calculamos o primeiro valor da box (explicamos como a seguir).

```
> Update(dt : float) : void
```

Como Minion anda em círculo, não em linha reta, não usamos uma velocidade com coordenadas x e y. Em vez disso, usamos uma velocidade angular (radianos/segundo), e incrementamos o arco a cada frame. Defina uma constante para isso, de preferência, uma fração de pi.

Calcular a posição do Minion é uma tarefa bem mais fácil do que parece. Começamos copiando a posição do centro do objeto orbitado. Como queremos percorrer uma circunferência, estaremos sempre a uma distância constante desse centro. Pensando nessa distância como a magnitude de um vetor, devemos criar uma componente x e uma componente y para esse vetor, e somá-las a posição do centro.

Essa projeção é exatamente a mesma que fizemos no movimento do Alien, tornada ainda mais simples porque já temos o ângulo calculado.

```
> Render() : void
```

Renderiza o minion na posição dada levando em conta o deslocamento da câmera.

```
> IsDead() : bool
```

Retorne false. Minion ser deletado não dependerá dele mesmo, mas sim do seu objeto pai.

Popule o vetor de Minions do Alien e observe o movimento. Tente mover pelo cenário e veja se eles acompanham o Alien corretamente.

```
> Shoot(pos : Point) : void
```

Shoot recebe uma posição e dispara um projétil naquela direção.

...espera, que projétil?

3. Bullet: Projétil Genérico

Bullet (herda de GameObject)
+ Bullet (x : float, y : float, angle : float, speed : float, maxDistance : float, sprite : string) + Update(dt : float) : void + Render() : void + IsDead() : bool
- sp : Sprite - speed : Point - distanceLeft : float

Bullet é um projétil que segue em linha reta após sua criação. Ele recebe vários parâmetros que a entidade atiradora determina, incluindo uma direção (angle), um módulo de velocidade (speed), um caminho para seu Sprite, e uma distância máxima a ser percorrida antes que o projétil "expire" (para que ele não ande pelo mundo infinitamente).

```
> Bullet (x : float, y : float, angle : float, speed : float,
          maxDistance : float, sprite : string)
```

Carregue o sprite passado e sete a box inicial. Daí, como Bullet tem a velocidade constante, calcule o vetor velocidade, pois este será usado em todo frame durante a vida do objeto. Lembre-se também de setar a distância remanescente de acordo com o parâmetro dado.

```
> Update(dt : float) : void
```

Para cada Update, a Bullet deve se mover speed * dt, e devemos subtrair essa mesma distância da distância remanescente.

```
> Render() : void
```

Renderiza sp na posição atual.

```
> IsDead() : bool
```

IsDead retorna true se já percorremos a distância máxima.

Temos um projétil pronto. Antes de fazermos alguém atirá-lo, no entanto, adicione o seguinte membro em State.

```
+ AddObject(GameObject* ptr) : void
```

Este método deve acrescentar o GameObject dado ao vetor de objetos. Voltemos ao Minion:

```
> Shoot(pos : Point) : void
```

Precisamos construir uma Bullet e acrescentá-la ao vetor de objetos. O projétil deve partir da posição do próprio Minion. O primeiro passo é calcular a direção (ângulo) que queremos que o projétil siga - novamente, é o mesmo cálculo para o movimento do Alien. O resto dos argumentos são constantes arbitrárias. O sprite é *img/minionBullet1.png*.

De volta em Alien: Se a ação a ser tratada é Shoot, escolha um Minion aleatório e mande-o disparar contra a posição dada. Se tudo estiver certo, a Bullet vai se mover na direção certa… Mas apontando para a direita.

Qual o seu problema, dona Bullet?

4. Sprites com Zoom e Rotação

Fizemos um objeto que gira em torno de outro, Minion, mas algo ainda mais interessante é podermos girar objetos em torno de si mesmos. Além disso, é comum querermos manipular a escala de objetos in-game, sem alterar seu sprite. Nossa engine não faz nada disso: vamos implementar! Adicione os seguintes membros em Sprite:

```
+ SetScaleX (scale : float) : void
+ SetScaleY (scale : float) : void

- scaleX : float
- scaleY : float
```

Inicialize as escalas como 1 em ambos os construtores de Sprite, para não quebrar os Sprites já no programa. Além disso, ajuste Sprite::GetWidth() e Sprite::GetHeight() para retornarem a dimensão ajustada para a respectiva escala. As funções SetScale são apenas métodos Set para a respectiva escala.

Agora precisamos ajustar a renderização. Um pequeno parêntese: Rotação e escala, na SDL 1.2, eram tarefas feitas por software por uma biblioteca um pouco temperamental, e davam brecha para vários bugs e memory leaks. Na SDL2, mudaremos exatamente três linhas do corpo de Sprite::Render para fazer o Sprite ser renderizado com zoom e/ou rotacionado.

Altere a assinatura do método para:

```
+ void Render(int x, int y, float angle = 0)
```

Para o zoom, você deve ajustar para a escala as dimensões do retângulo de destino (quarto argumento da SDL_RenderCopy). O tamanho do Sprite será ajustado automaticamente para ocupar o novo retângulo.

Para a rotação, vamos substituir a SDL_RenderCopy pela SDL_RenderCopyEx. Ela recebe sete argumentos, sendo os quatro primeiros os mesmos da RenderCopy. Os três outros são:

- `angle : double` - Ângulo de rotação no **sentido horário** em **graus**.
- `center : SDL_Point*` - Determina o eixo em torno da qual a rotação ocorre. Se passarmos NULL, a rotação ocorre em torno do centro do retângulo de destino, que é o que queremos.
- `flip : SDL_RendererFlip` - Inverte a imagem verticalmente (SDL_FLIP_VERTICAL), horizontalmente (SDL_FLIP_HORIZONTAL), ambos (bitwise or), ou não inverte (SDL_FLIP_NONE). Você pode implementar suporte à inversão de Sprites se quiser, mas por enquanto, use SDL_FLIP_NONE.

Pronto! Com isso, basta setar as escalas e rotações em seus objetos. Adicione o seguinte membro em GameObject…

```
+ rotation : float
```

Esse será o ângulo de rotação do corpo. Recomenda-se criar um construtor padrão para GameObject que inicialize a variável como zero. Sempre que quiser que um objeto gire na tela, mantenha o ângulo de rotação do corpo nessa variável, e passe para a chamada à renderização do Sprite.

Para esse exercício, faça:

1. Minions sempre com a parte de baixo do Sprite virada para o Alien;
2. Alien girando lentamente na direção contrária à translação dos Minions;
3. Minions com escala aleatória entre 1 e 1.5.

E por hoje, é só. Lembre-se de mover a câmera para testar o trabalho completo.

+ Extra (+0,5 ponto) : Faça com que o Minion mais próximo da posição onde o clique ocorreu seja escolhido para atirar.