Desenvolvimento de Jogos

Agenda

- Panorama Geral
  - Histórico e contextualização
  - Mercado
  - Ferramentas e linguagens
- Desafio Wumpus
  - Contextualização
  - Aplicação em funcionamento
- Considerações

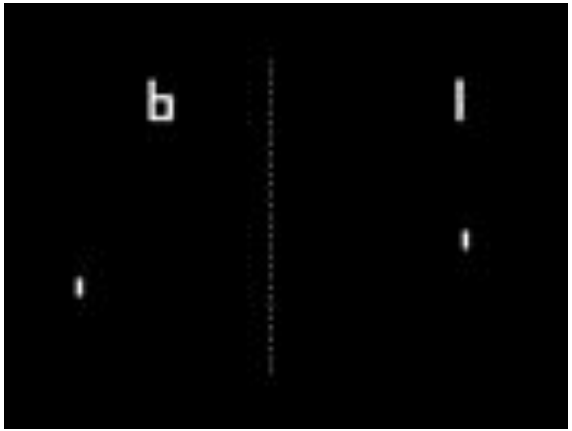
Debora Cristina Fontanella

Panorama geral - Histórico

Primeiros Jogos (anos 50)

O primeiro jogo foi um Jogo da Velha, desenvolvido em 1952 para uma tese de doutorado. Apesar de inovadores pobres de estrutura e elaborados sem abordagens sistemáticas, quando comparados aos produtos mais modernos.

Desenvolvidos inicialmente como estudo em ciência da computação para estudos de interação homem-máquina, inteligência artificial e simuladores.



Interface do Pong. Primeiro jogo desenvolvido pela Atari.

Panorama geral - Histórico

ERA 8-BIT

Os primeiros jogos eram de 8bits e eram desenvolvidos usando Assembler + Qbasic.

Qbasic foi a linguagem base de programação para vários consoles e computadores. Havia um console extra (Development Kit) com uma saída RGB-DV que passava os códigos pŕe-compilados para uma bios de teste e nos primeiros consoles como Atari. Cada nova compilação sobrescrevia a bios. Os jogos eram testados no computador, passados para a bios do console para ver se aguentava, e depois eram gravados os cartuchos com o game

Principais computadores: Apple IIc, Atari ST e Commodore 64

Exemplo de dev Kit

Exemplo de dev Kit

Exemplo de dev Kit

Panorama geral - Histórico

ERA 16-BIT

Computadores mais modernos. Ex: Next Computer, usando programação em Objective-C, orientado a objetos. Nesse computador foi feito o primeiro DOOM, também Quake. Foi muito usado para programação 32 bits também. Os efeitos visuais de Jurassic Park e Toy Story foram feitos usando NextStep

Nessa época utilizava-se bastante programação C.

Outra ferramenta: Allegro (biblioteca)

Panorama geral - Histórico

O Kit de Desenvolvimento do PC Engine e Turbo Grafix 16 chamados de (PC Engine PDS Dev-Kit) eram ligados no Next Computer que tinha um sistema operacional com vários programas para desenvolvimento, o sistema operacional desse PDS era o Nextstep o mesmo do Next Computer.

Next Computer


94
AMMO
52%
HEALTH
2 3 4
5 6 7
ARMS
115%
ARMOR
BULL 29 / 400
SHEL 42 / 100
RCKT 10 / 100
CELL 94 / 600
trial version

Doom 1

Quake

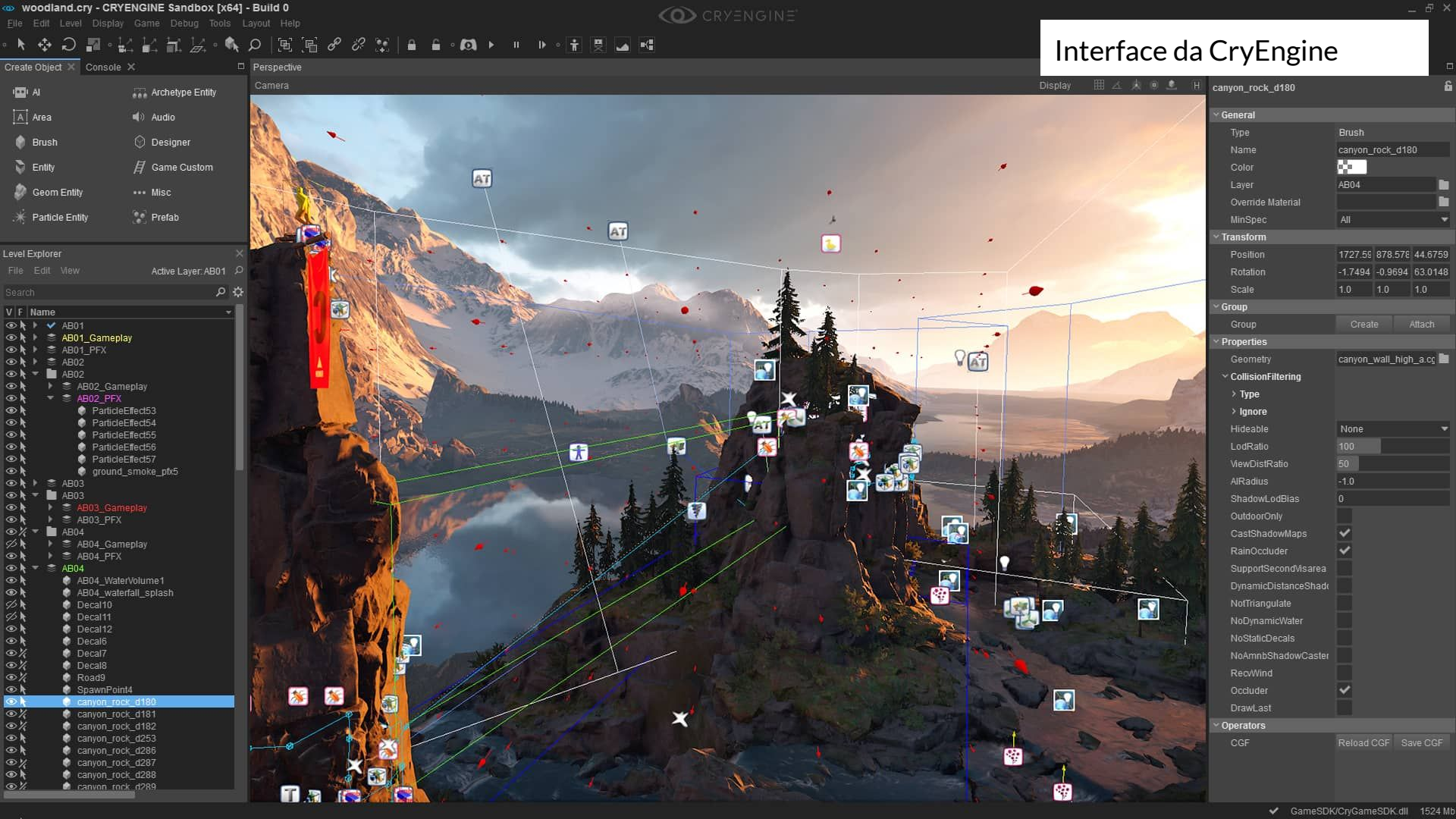
Panorama geral - Histórico

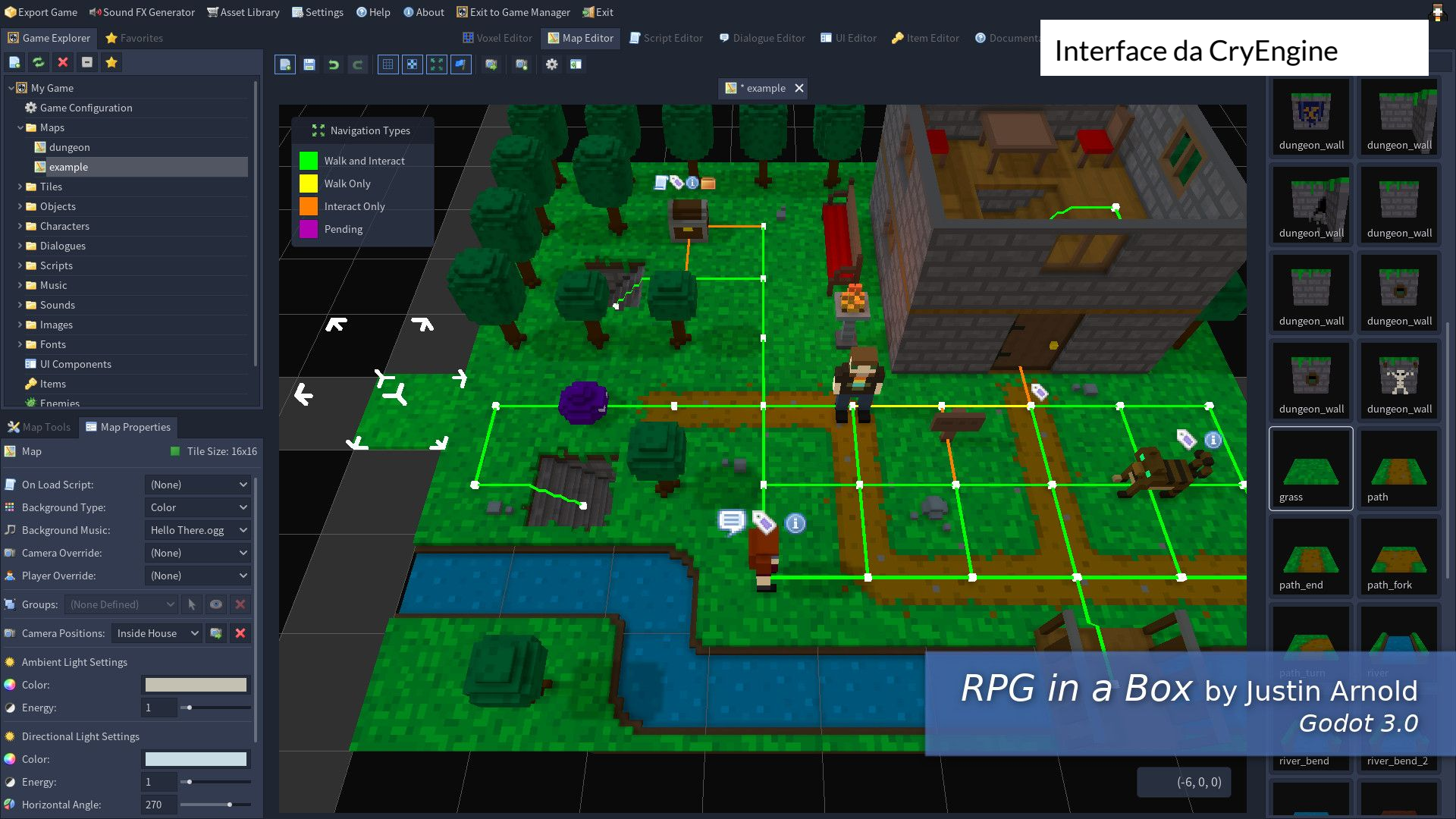
- Anos 50
  - Primeiros Jogos (fins acadêmicos)
- Anos 60
  - Primeiro jogo doméstico
- Anos 70
  - Lançamento do Atari (Sega)
  - Jogos arcade (máquinas)
    - Space invaders, Pong, Donkey Kong, Pac-man
  - Primeiros computadores pessoais
- Anos 80
  - Primeiras revistas de games (comunidade gamer)
  - Consoles:
    - Super Nintendo, Sega Mega Drive, Gameboy
- Anos 90
  - Nintendo 64, Playstation, Sega Saturn
  - Utilização de componentes de computador (processador, memória RAM, placa de vídeo...) na produção dos consoles. Playstation trouxe o cartão de memória
  - Começo dos jogos em 3D e CDs
  - Primeiros jogos online
    - FPS (Quake)
    - RPGs (Warcraft, Starcraft...)
- Anos 2000 até atualidade
  - Estabelecimento de jogos mobile
  - Investimento em Realidade Virtual
  - E-sports
  - Sensores de movimento (nintendo Wii, Xbox Kinect)

Panorama Geral - Mercado, Ferramentas e Linguagens

- Áreas: entretenimento, meio científico, jogos sob encomenda, gameficação (essas duas últimas vindas principalmente da área de publicidade e marketing)
- Raramente um jogo é desenvolvido por um único indivíduo, envolve uma equipe interdisciplinar que inclui produtores, designers, programadores, artistas, roteiristas, engenheiros de som e testadores.
- Engines mais populares: Unity (mobile), Unreal Engine (Playstation), CryEngrine, Source e Radiant.
- Linguagens: C++, Lua, Java (multiplataforma); HTML, CSS e JavaScript (web ou mobile)


Bike_Demo
File
Edit
Asset
Window
Help
Save
Browse
Import
Execute
Commit
Interface da Unreal Engine
Content
Demo
Front_plastic_x_-_lightr
Geometries
Front_plastic_x_-_lig
Materials
Master
M_DatasmithCAD
color_6179c9ff
color_d6d6ebff
Gas_tank_-_lightning_c
Geometries
Gas_tank_-_lightnin
Materials
Master
M_DatasmithCAD
color_6179c9ff
color_d6d6ebff
Unreal_Sportbike_SLDA
Geometries
Perspective
Lit
Show
Rendering
Current Screen Size: 1,670149
#Static Meshes: 601
#Meshes drawn: 2,015
#Vertices Used: 4,012,289
Approx Size: 82x214x122
Label
Type
Preview
World
Front_plastic_x_-_Datasmith
Front_plastic_x_Actor
Front_plastic_StaticMesh
Gas_tank_-_lightn
Gas_tank_-_ligh
Gas_tank_-_li
Unreal_Sportbike
Unreal_Sportbik
Cable_1
Actor
Clutch_cab
Clutch_c
Clutch_c
Cooler_injec
Cooler_in
Cooler_injec
Inputs
Filename
C:/Users/Owner/Document
Output
Folder
/Content/Demo/
Sub-Level
Bike
Search Details
Transform
Location
Rotation
Scale
Mobility
Static Mesh
Static Mesh
Materials
Element 0
Textures
Physics
Select By
Condition
Is Class Of
Float
Bounding Volu
String
Actor Label
Actor Tag
Metadata Value
Object Name
Operations
On Actor
Compact Scene
Dataprep Graph
Zoom -3
Tessellation
Assign Materials
Cleanup
Datasmith Tessellation
Chord Tolerance
Max Edge Length
Normal Tolerance
Set Mobility
Mobility Type
Movable
Set Simple Collision
Substitute Material By Table
Material Data Table
DATAPREP


woodland.cry - CRYENGINE Sandbox [x64] - Build 0
File Edit Level Display Game Debug Tools Layout Help
CRYENGINE
Interface da CryEngine
Create Object
Console
AI
Archetype Entity
Area
Audio
Brush
Designer
Entity
Game Custom
Geom Entity
Misc
Particle Entity
Prefab
Level Explorer
File Edit View
Active Layer: AB01
Search
V F Name
AB01
AB01_Gameplay
AB01_PFX
AB02
AB02_Gameplay
AB02_PFX
ParticleEffect53
ParticleEffect54
ParticleEffect55
ParticleEffect56
ParticleEffect57
ground_smoke_pfx5
AB03
AB03_Gameplay
AB03_PFX
AB04
AB04_Gameplay
AB04_PFX
AB04_WaterVolume1
AB04_waterfall_splash
Decal10
Decal11
Decal12
Decal6
Decal7
Decal8
Road9
SpawnPoint4
canyon_rock_d180
canyon_rock_d181
canyon_rock_d182
canyon_rock_d253
canyon_rock_d286
canyon_rock_d287
canyon_rock_d288
canyon_rock_d289
Perspective
Camera
Display
canyon_rock_d180
General
Type Brush
Name canyon_rock_d180
Color
Layer AB04
Override Material
MinSpec All
Transform
Position 1727.56 878.576 44.6759
Rotation -1.7494 -0.9694 63.0148
Scale 1.0 1.0 1.0
Group
Group Create Attach
Properties
Geometry canyon_wall_high_a.cg
CollisionFiltering
Type
Ignore
Hideable None
LodRatio 100
ViewDistRatio 50
AIRadius -1.0
ShadowLodBias 0
OutdoorOnly
CastShadowMaps
RainOccluder
SupportSecondVisarea
DynamicDistanceShadc
NotTriangulate
NoDynamicWater
NoStaticDecals
NoAmnbShadowCaster
RecvWind
Occluder
DrawLast
Operators
CGF Reload CGF Save CGF
GameSDK/CryGameSDK.dll 1524 Mb


Export Game
Sound FX Generator
Asset Library
Settings
Help
About
Exit to Game Manager
Exit
Game Explorer
Favorites
Voxel Editor
Map Editor
Script Editor
Dialogue Editor
UI Editor
Item Editor
Documenta
Interface da CryEngine
My Game
Game Configuration
Maps
dungeon
example
Tiles
Objects
Characters
Dialogues
Scripts
Music
Sounds
Images
Fonts
UI Components
Items
Enemies
* example
Navigation Types
Walk and Interact
Walk Only
Interact Only
Pending
dungeon_wall
grass
path
path_end
path_fork
river_bend
river_bend_2
Map Tools
Map Properties
Map
Tile Size: 16x16
On Load Script:
(None)
Background Type:
Color
Background Music:
Hello There.ogg
Camera Override:
Player Override:
Groups:
(None Defined)
Camera Positions:
Inside House
Ambient Light Settings
Color:
Energy:
1
Directional Light Settings
Horizontal Angle:
270
(-6, 0, 0)
RPG in a Box by Justin Arnold
Godot 3.0

Mundo Wumpus - Contextualização

O Mundo de Wumpus é um jogo antigo de computador composto por um ambiente artificial que fornece grande motivação para o raciocínio lógico. Trata-se de um excelente ambiente de teste para agentes inteligentes (inteligência artificial) pois trabalha com conhecimento e descoberta.

Baseia-se em um agente que deve explorar uma caverna, composta por compartimentos pelos quais o agente pode se mover. Nos compartimentos podem existir buracos em que o agente cai, ou monstros (Wumpus) que devoram o agente. A posição inicial do agente é [1,1] e seu objetivo é encontrar o ouro e voltar à posição inicial (saída da caverna) sem cair em um buraco ou ser devorado pelo Wumpus. Além disso, o agente possui uma flecha disponível, que pode atirar em uma direção (direita, esquerda, acima ou abaixo) para tentar matar o Wumpus.

Fonte: https://www.ime.usp.br/~leliane/IAcurso2000/Wumpus.html

As ações possíveis são:

- Mover-se para o próximo compartimento em uma direção
- Atirar flecha em uma direção

Os movimentos são limitados às direções horizontais e verticais, nunca na diagonal, da mesma forma os elementos de percepção do agente. As percepções do agente (aquisição de conhecimento) são o que guiam seus movimentos. Em cada compartimento ele pode encontrar:

- BRISA - Indica que há um buraco em um dos compartimentos adjacentes
- FEDOR - Indica que há um Wumpus em um dos compartimentos adjacentes
- BRILHO - Indica que há ouro no compartimento visitado

Em qualquer lugar da caverna, pode perceber GRITO - Sinal de que o Wumpus foi atingido pela flecha e morreu.

Fonte: https://www.ime.usp.br/~leliane/IAcurso2000/Wumpus.html

Mundo Wumpus - Construção

Ferramentas:

VueJs (framework JavaScript), HTML, CSS. Motivo: Maior afinidade, economia de tempo de aprendizado de uma engine específica para jogos. VueJs facilita na animação

SEQUÊNCIA
- Construir o cenário estático
- Construir cenário de forma dinâmica
- Ações do agente + Feedback das ações

CONTROLES
- Mover o agente - teclas de direção
- Disparar flecha em uma direção - W,A,S,D
- Pegar o ouro - Barra de espaço

Cenário estático
- Tabela (caverna) - (fundo: textura de terra)
- Células (compartimento) - Fundo Fedor ou Vento, conforme posição do buraco e do Wumpus
- Agente, Wumpus, Ouro e Buraco: Componente com imagem

(Elementos gráficos encontrados no Google Imagens)

Cenário dinâmico:
- Arrays para: posições de buracos, posições de wumpus, posições de ouro
- Objeto: posição do agente
- Gerar tabela a partir de matriz [n,m] - Para cada célula, função que identifica se há um wumpus ou buraco e coloca os sinais nas posições adjacentes. Bem como o wumpus, ouro, agente e buraco nas posições indicadas

Ações do Agente

- O Agente é um componente encapsulado que tem como função executar as ações e enviar o sinal para o tabuleiro, que conhece a posição dos demais elementos:
    - exibição com/sem flecha
    - disparar flecha (envia direção para a qual a flecha foi atirada e remove flecha do agente)
    - andar (envia para o tabuleiro a nova posição do agente)
    - pegar ouro (envia ação para o tabuleiro que identifica se há ouro na casa)

Alertas:

A cada ação, além de descontar 1 ponto disparada pelo agente verifica-se:

- Andar: se caiu em um buraco ou foi comido pelo wumpus (envia um alerta e desconta pontos, - 1000 por ser comido pelo wumpus, -20 por cair no buraco) / se percebe brisa/fedor e envia alerta de que há um dos respectivos elementos por perto.
- Atirar flecha: se a flecha atingiu ou não o wumpus (+ 100)
- Pegar ouro (se há ouro na posição do agente, pega o ouro e adiciona pontos, + 200)
- Vencer jogo - se chegou à casa inicial, com o ouro coletado

Considerações:

1. Se um jogo simples já exigiu bastante trabalho, imagina um jogo elaborado. Ainda bem que atualmente existem engines (e mesmo assim é ultra trabalhoso), devia ser muito difícil desenvolver um jogo com ferramentas menos avançadas.
2. Por ser um jogo simples, valeu mais a pena investir tempo em construção de ações básicas (que já existem em engines) do que aprender uma ferramenta. Mas se houvesse mais tempo hábil e a intenção fosse aprender uma ferramenta, seria mais vantajoso, definitivamente.
3. Desenvolver um jogo envolve muito mais detalhes do que uma aplicação simples
4. Foi um trabalho diferente, porém bastante trabalhoso
5. Opinião extremamente pessoal: O papel da computação dentro do desenvolvimento de games é apenas no que diz respeito ao hardware, softwares auxiliares e à programação = a área de TI não é nada sem outras áreas do conhecimento. Profissionais de TI não são super-heróis
6. Aplicação disponibilizada em: https://github.com/fcristinadebora/wumpus-game
7. Projeto usado como base: https://github.com/joaopandolfi/Wumpus