Simulador de um Ambiente Virtual Distribuído Multiusuário para Batalhas de Tanques 3D com Inteligência Baseada em Agentes BDI

Acadêmico: Germano Fronza
gfronza@inf.furb.br

Orientador: Dalton Solano dos Reis
dalton@furb.br

julho de 2008

Roteiro da Apresentação

1. **Introdução:** Contextualização e Objetivos
2. **Fundamentação Teórica**
   (a) AVD, SMA, JME
   (b) Trabalhos Correlatos
3. **Desenvolvimento do Simulador**
   (a) Principais Requisitos
   (b) Especificação
   (c) Implementação
   (d) Resultados e Discução
4. **Conclusão e Extensões**

Introdução

Contextualização

- Responsabilidade dos simuladores.
- Ambientes 3D e o aumento da imersão nos simuladores.
- Usuários remotos $\longrightarrow$ o surgimento dos AVDs.
- IA contribuindo para o aumento da realidade nos simuladores.
- Simulador provendo um ambiente atrativo para desenvolvimento e testes de agentes BDI.

Objetivos do Trabalho

- Desenvolver um simulador 3D de um AVD para atuar em uma rede local.
- Objetivos específicos:
  - ⋆ Definir um cenário a ser utilizado no simulador (ambiente, objetos, percepções e comportamentos);
  - ⋆ Utilizar a engine gráfica JMonkey Engine para construção do ambiente gráfico do AVD;
  - ⋆ Permitir que os tanques sejam controlados por avatares ou tenham um comportamento autônomo.

Fundamentação Teórica

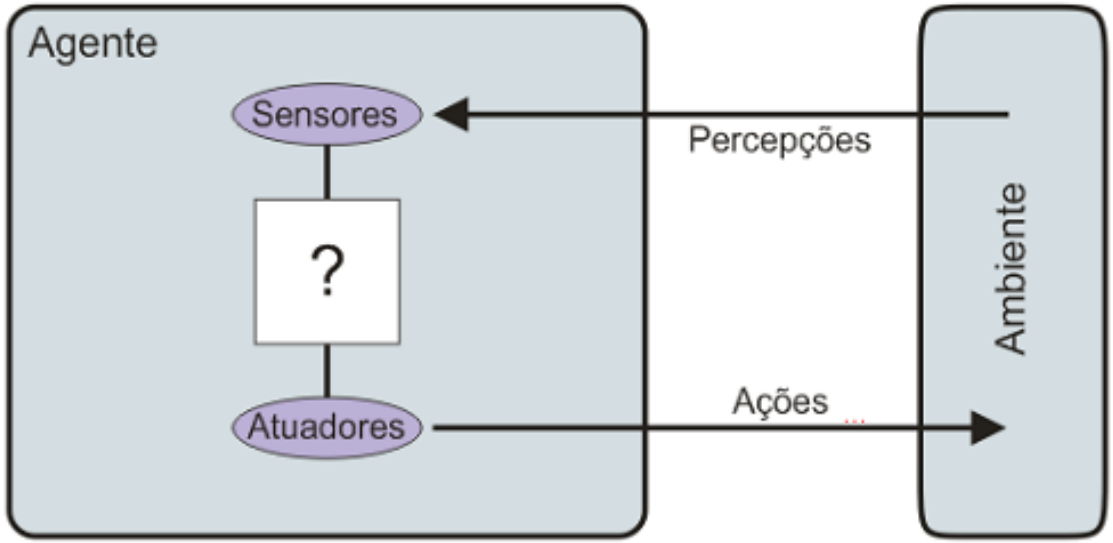
Ambiente Virtual Distribuído

- Primeiros AVDs desenvolvidos:
  - ⋆ Simulator Network (SIMNET);
  - ⋆ Naval Postgraduate School Network (NPSNET).
- Resultados obtidos a partir desses projetos.
- Modelos gerais de comunicação:
  - ⋆ Centralizado;
  - ⋆ Distribuído.

- Java Game Networking (JGN)
  - ⋆ *Open-source*.
  - ⋆ Maior **abstração** na implementação do módulo de comunicação.
  - ⋆ Mensagens definidas como **classes Java**.
  - ⋆ Faz uso dos protocolos TCP e UDP.
  - ⋆ Suporta UDP *broadcasting*.
  - ⋆ Falta de documentação $\longrightarrow$ ponto negativo.

Sistemas MultiAgentes (SMA)

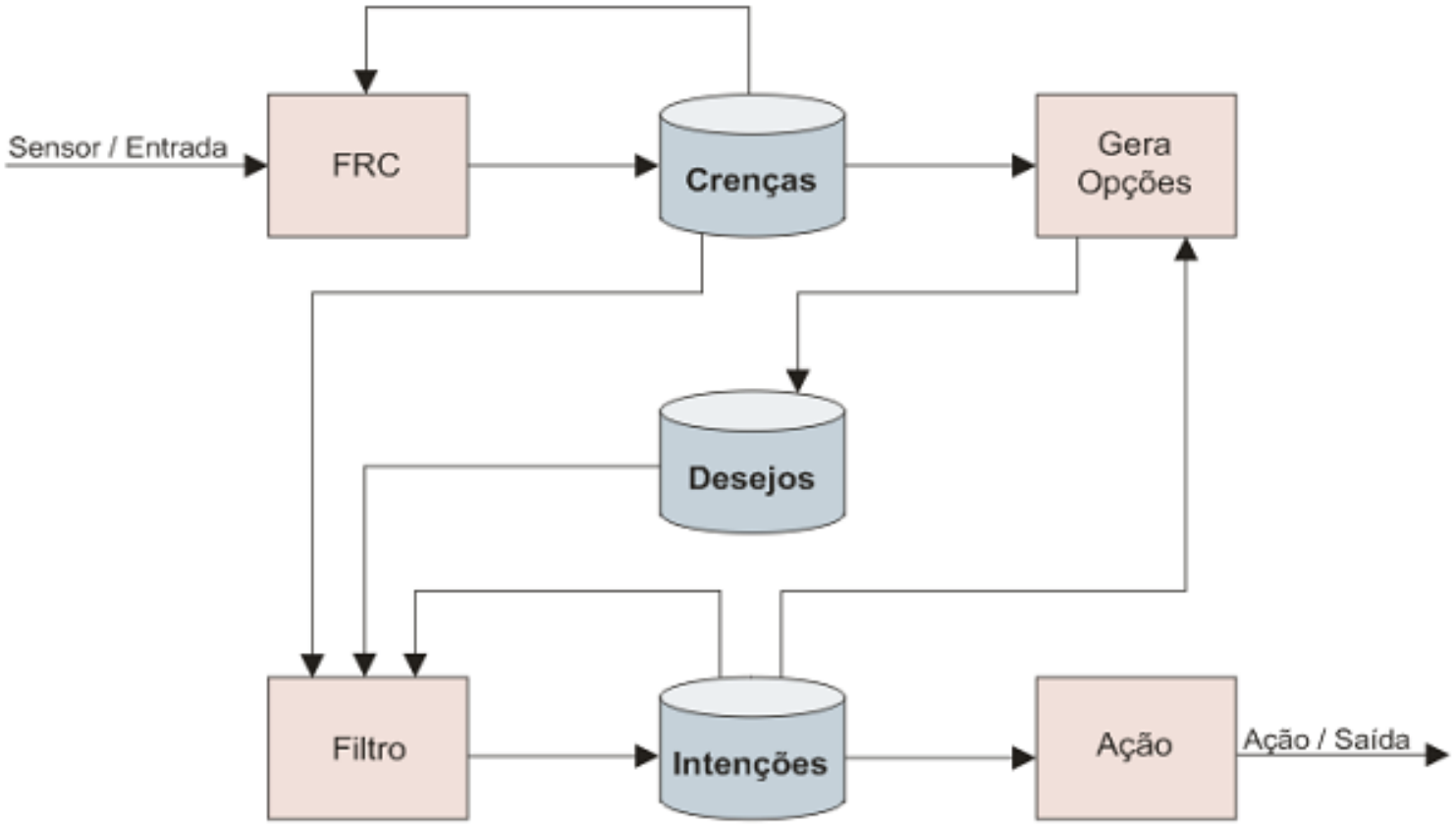
"Os **sistemas multiagentes** são sistemas compostos por multiplos elementos computacionais interativos, conhecidos como **agentes**." (WOOLDRIDGE, 2002)



- Autonomia, Pró-atividade, Reatividade e Habilidade social.

Arquitetura BDI

- Atitudes mentais.
- *Belief, Desire and Intentions* (BDI).



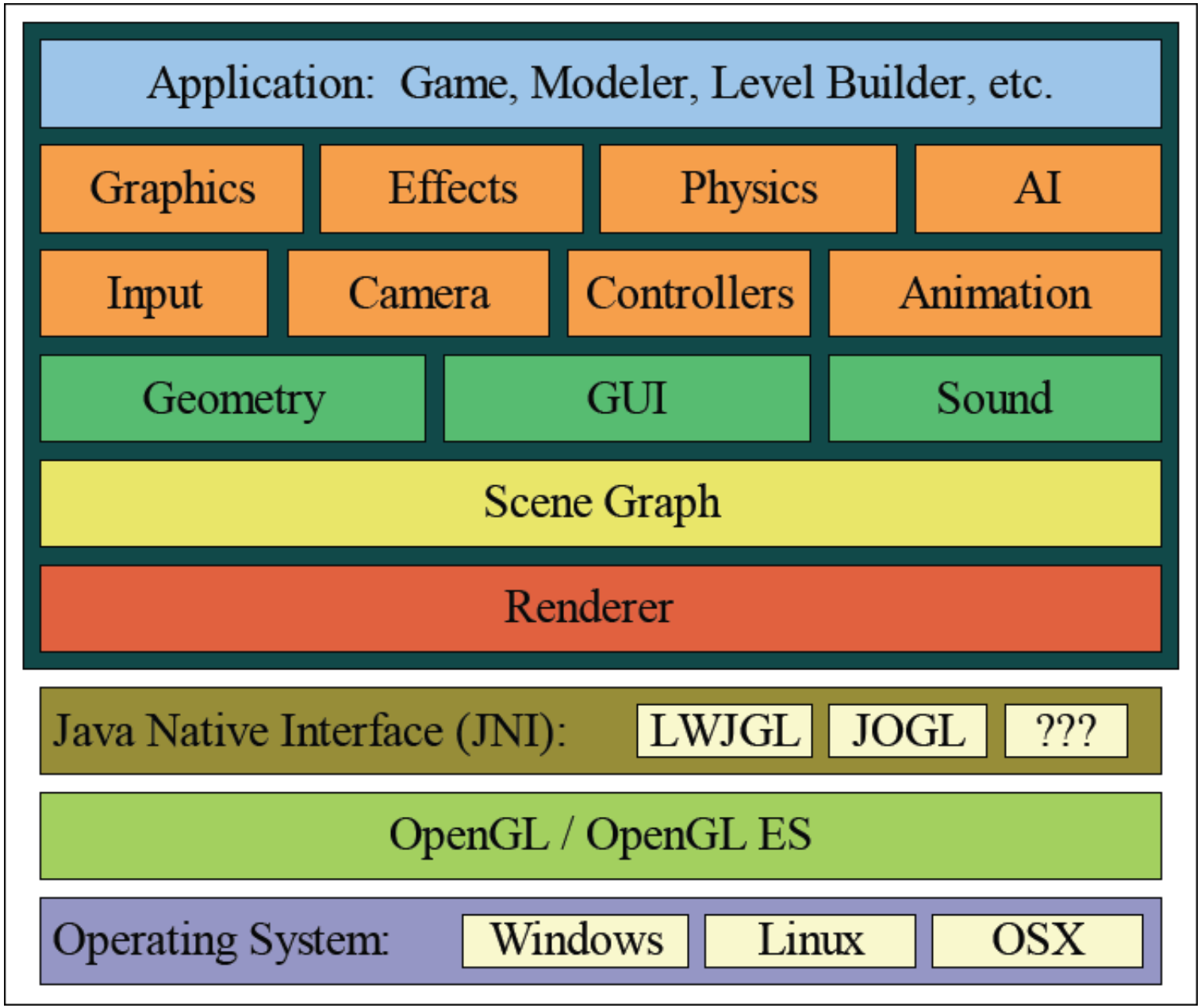
Linguagem AgenteSpeak(L) e o interpretador Jason

- Sintaxe abstrata.
- Linguagem de lógica, semelhante ao Prolog.
- Código de um agente AgentSpeak(L) é composto por um conjunto de crenças iniciais, metas e planos.

```
1
2   /** Crenças iniciais */
3   likes(orquestraSinfonica).
4   likes(robertoCarlos).
5   likes(coralDeNatal).
6
7   /** Planos */
8   +concert(A, V) : likes(A)
9        ← !book_tickets(A, V).
10
11  +!book_tickets(A, V) : ¬busy(phone)
12       ← call(V);
13         ...;
14         !choose_seats(A, V).
15
```

JMonkey Engine (JME)

- *Open-source*.
- Escrita totalmente em Java delegando tarefas de renderização à biblioteca nativa OpenGL.
- Abstrai e encapsula tarefas de baixo nível em classes Java.
- Baseada em Grafo de Cena.
- Sem acoplamento com uma implementação especifica de OpenGL para Java (através de JNI)
- Física usando o *framework* JME Physics.
- Conceito de *Game States*.


Application: Game, Modeler, Level Builder, etc.
Graphics
Effects
Physics
AI
Input
Camera
Controllers
Animation
Geometry
GUI
Sound
Scene Graph
Renderer
Java Native Interface (JNI):
LWJGL
JOGL
???
OpenGL / OpenGL ES
Operating System:
Windows
Linux
OSX

Trabalhos Correlatos

- Robocode.
- Sistema multiagentes utilizando a linguagem AgentSpeak(L) para criar estratégias de armadilha e cooperação em um jogo tipo PacMan.
- Protótipo de um ambiente virtual distribuído multiusuário.

Desenvolvimento do Simulador

Principais Requisitos

Funcionais:

- Permitir a escolha do mapa da batalha.
- Possibilitar a entrada de novos modelos de mapas.
- Disponibilizar os dois modelos de tanques de guerra.
- Fornecer um conjunto de ações, percepções e crenças para os agentes.
- Permitir que os tanques sejam controlados por usuários ou tenham um comportamento autônomo.
- Prover um AVD com arquitetura híbrida para LAN.

Não-Funcionais:

- Tratar no máximo seis tanques de guerra em cada time.
- Garantir que nenhum usuário se conecte ao servidor após o início da batalha.
- JavaSE 6.0, JMonkey Engine, JME Physics, JGN e Jason.

Especificação

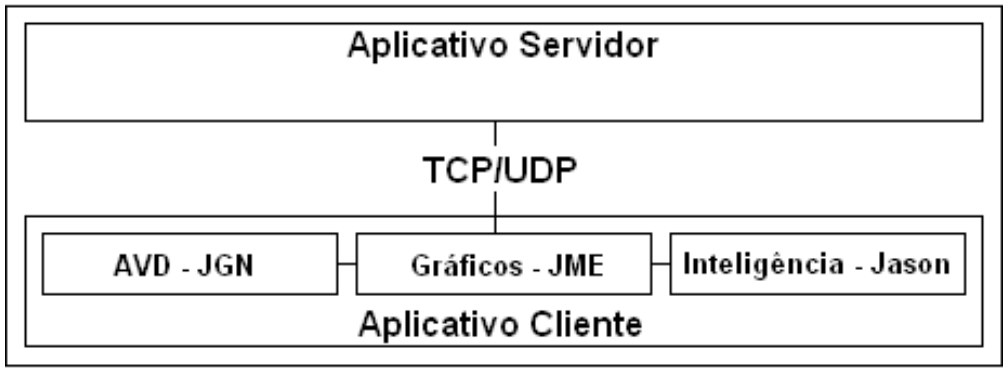
Técnicas e Ferramentas Utilizadas

- Ferramenta StarUML:
  - ⋆ *Freeware*;
  - ⋆ *Open-source*;
  - ⋆ Geração de código e implementação de padrões (Design Patterns - GoF);
- Usada para criar os diagramas da UML:
  - ⋆ Classes;
  - ⋆ Estados;
  - ⋆ Grafo de Cena.

Especificação da Arquitetura do Simulador

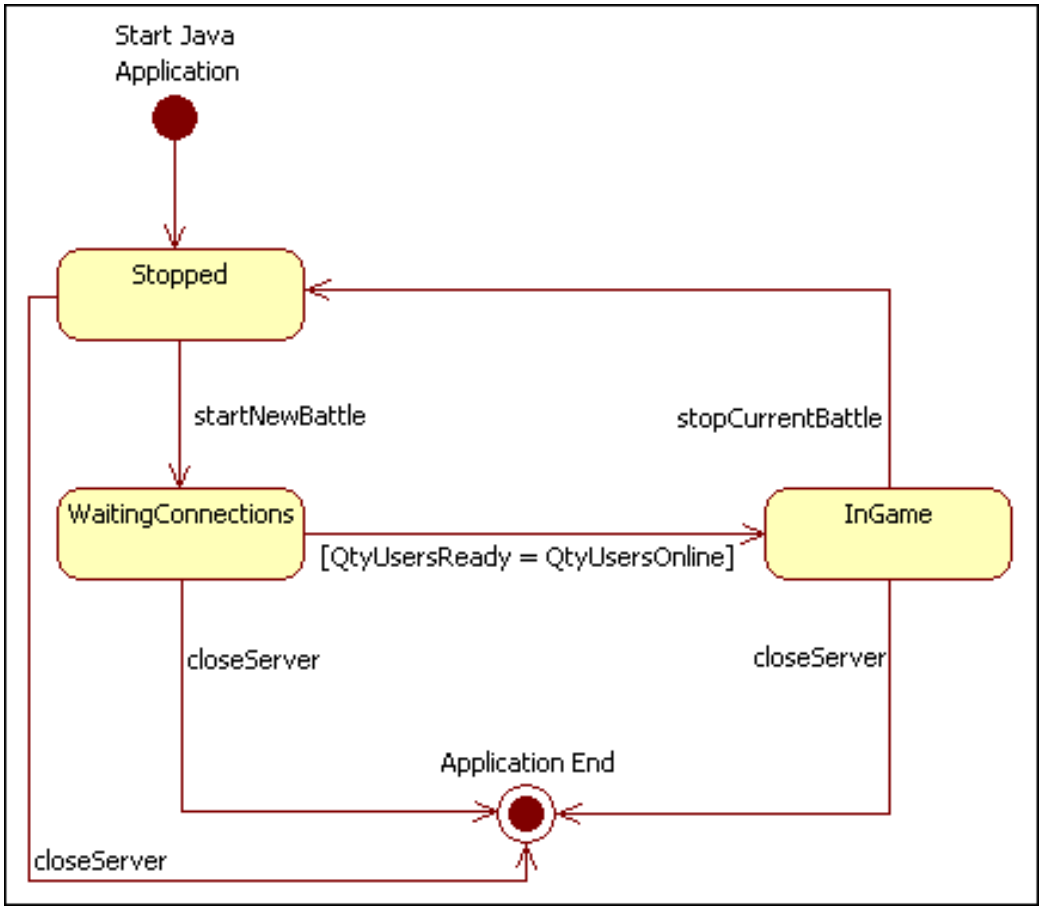
Arquitetura Híbrida: Características de arquitetura centralizada combinadas com características de arquitetura distribuída.

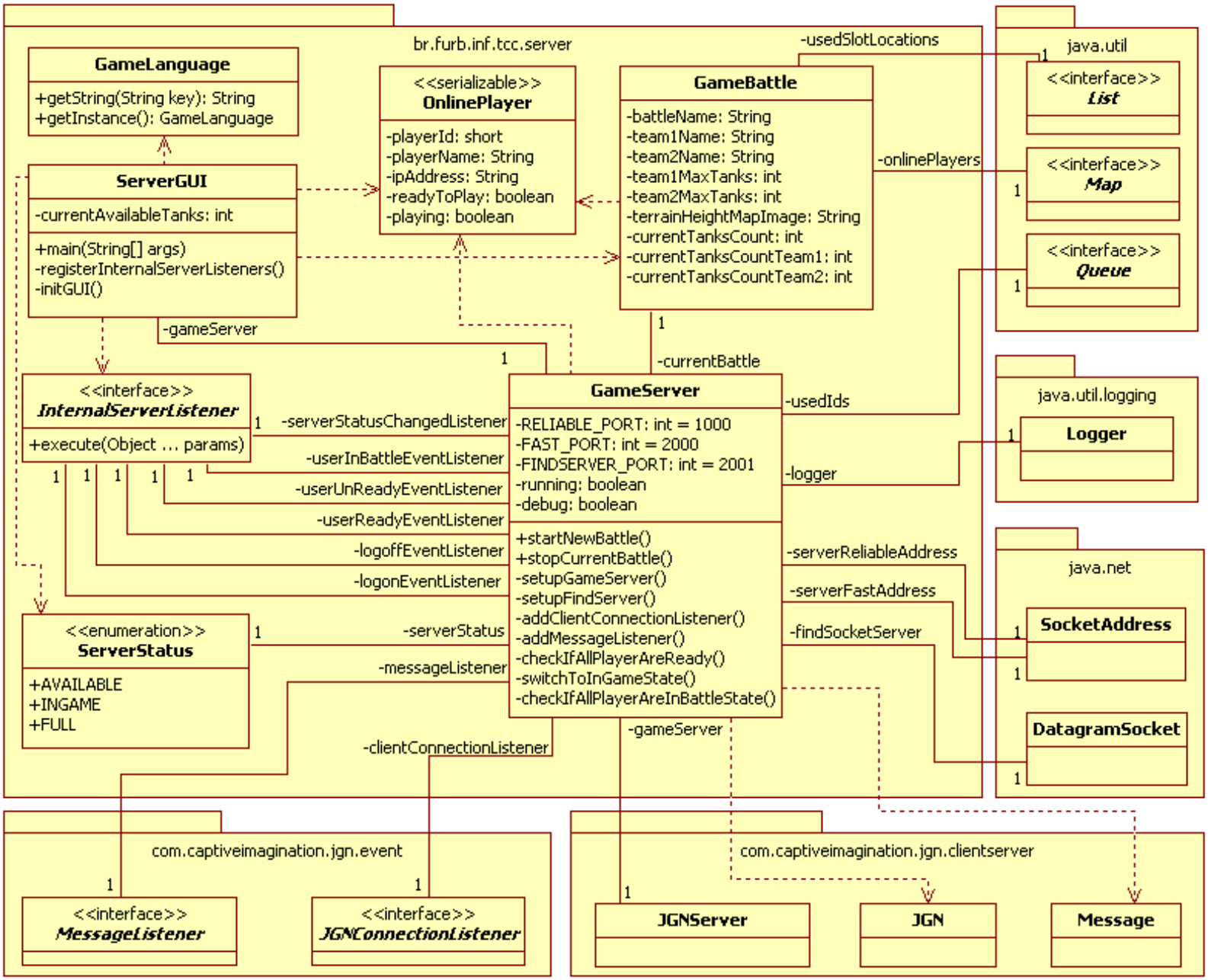
- O Aplicativo Servidor;
- O Aplicativo Cliente.



O Aplicativo Servidor

Estados possíveis:



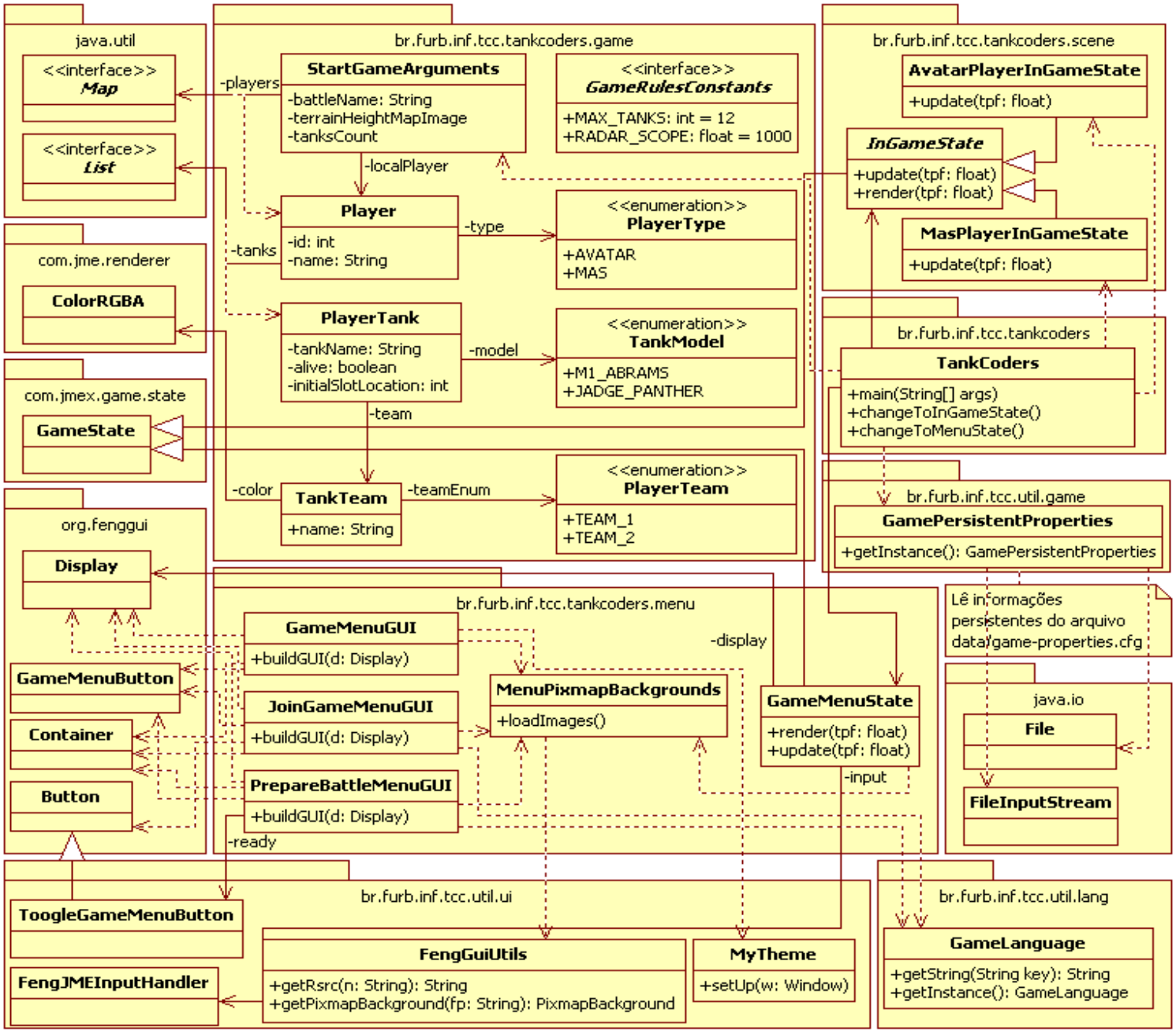

br.furb.inf.tcc.server
GameLanguage
+getString(String key): String
+getInstance(): GameLanguage
ServerGUI
-currentAvailableTanks: int
+main(String[] args)
-registerInternalServerListeners()
-initGUI()
<<serializable>>
OnlinePlayer
-playerId: short
-playerName: String
-ipAddress: String
-readyToPlay: boolean
-playing: boolean
GameBattle
-battleName: String
-team1Name: String
-team2Name: String
-team1MaxTanks: int
-team2MaxTanks: int
-terrainHeightMapImage: String
-currentTanksCount: int
-currentTanksCountTeam1: int
-currentTanksCountTeam2: int
-usedSlotLocations
-onlinePlayers
-gameServer
-currentBattle
<<interface>>
InternalServerListener
+execute(Object ... params)
-serverStatusChangedListener
-userInBattleEventListener
-userUnReadyEventListener
-userReadyEventListener
-logoffEventListener
-logonEventListener
GameServer
-RELIABLE_PORT: int = 1000
-FAST_PORT: int = 2000
-FINDSERVER_PORT: int = 2001
-running: boolean
-debug: boolean
+startNewBattle()
+stopCurrentBattle()
-setupGameServer()
-setupFindServer()
-addClientConnectionListener()
-addMessageListener()
-checkIfAllPlayerAreReady()
-switchToInGameState()
-checkIfAllPlayerAreInBattleState()
-usedIds
-logger
-serverReliableAddress
-serverFastAddress
-findSocketServer
<<enumeration>>
ServerStatus
+AVAILABLE
+INGAME
+FULL
-serverStatus
-messageListener
-clientConnectionListener
-gameServer
java.util
<<interface>>
List
<<interface>>
Map
<<interface>>
Queue
java.util.logging
Logger
java.net
SocketAddress
DatagramSocket
com.captiveimagination.jgn.event
<<interface>>
MessageListener
<<interface>>
JGNConnectionListener
com.captiveimagination.jgn.clientserver
JGNServer
JGN
Message

O Aplicativo Cliente

- Representação gráfica do estado do AVD.
  - ⋆ *Middleware* Java Game Networking (JGN).
- Interação com o aplicativo servidor.
  - ⋆ *Engine* gráfica JMonkey Engine (JME).
- Integração com o interpretador Jason.
  - ⋆ Interpretador Jason.

Representação gráfica do estado do AVD

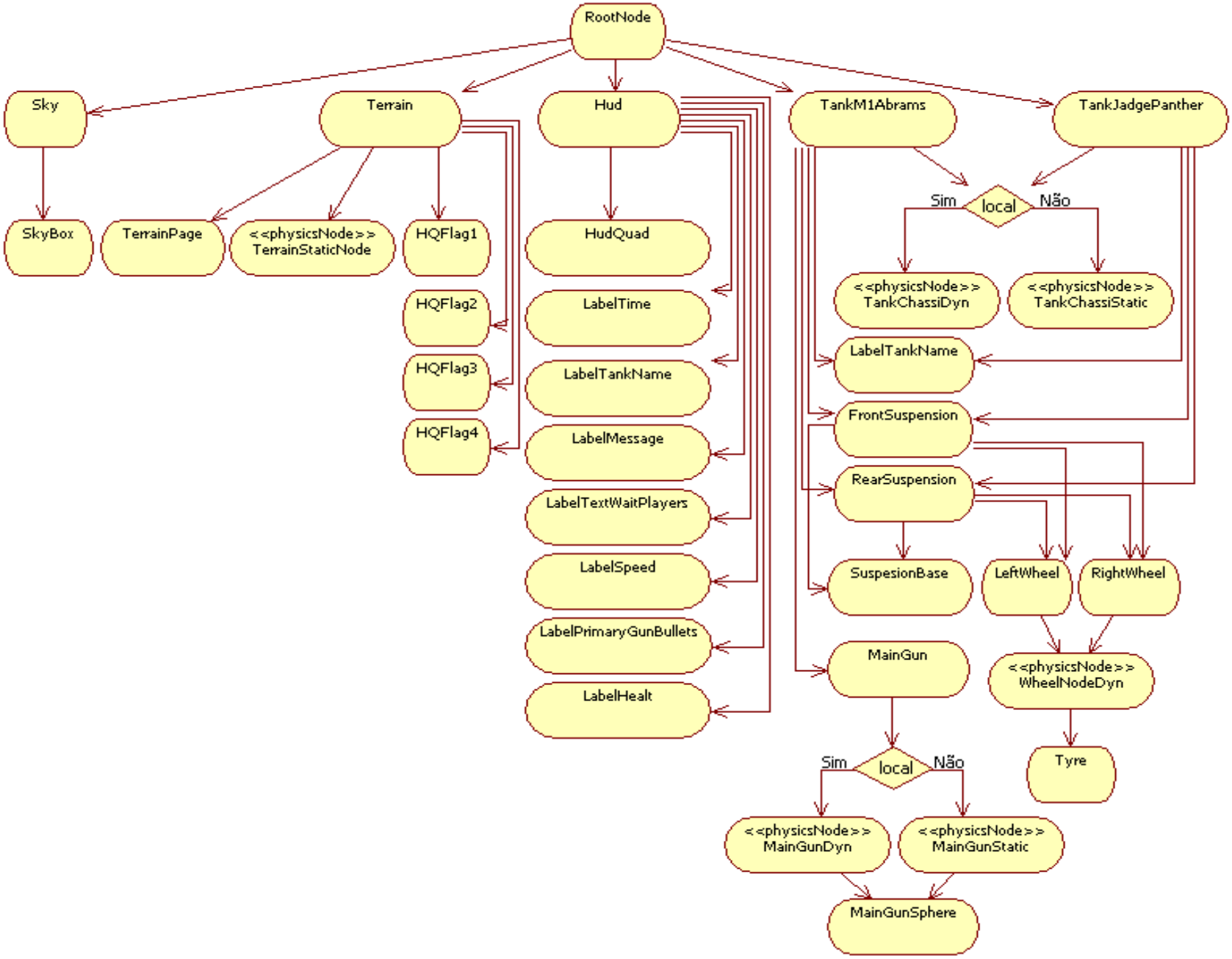
- *Game States* do simulador:
  - ⋆ `GameMenuState`;
  - ⋆ `InGameState`.
- Somente um *game state* ativo no `GameStateManager`.


java.util
<<interface>>
Map
<<interface>>
List
com.jme.renderer
ColorRGBA
com.jmex.game.state
GameState
org.fenggui
Display
GameMenuButton
Container
Button
ToogleGameMenuButton
FengJMEInputHandler
br.furb.inf.tcc.tankcoders.game
StartGameArguments
-battleName: String
-terrainHeightMapImage
-tanksCount
-players
-localPlayer
<<interface>>
GameRulesConstants
+MAX_TANKS: int = 12
+RADAR_SCOPE: float = 1000
Player
-id: int
-name: String
-tanks
-type
<<enumeration>>
PlayerType
+AVATAR
+MAS
PlayerTank
-tankName: String
-alive: boolean
-initialSlotLocation: int
-model
<<enumeration>>
TankModel
+M1_ABRAMS
+JADGE_PANTHER
-team
-color
TankTeam
+name: String
-teamEnum
<<enumeration>>
PlayerTeam
+TEAM_1
+TEAM_2
br.furb.inf.tcc.tankcoders.scene
AvatarPlayerInGameState
+update(tpf: float)
InGameState
+update(tpf: float)
+render(tpf: float)
MasPlayerInGameState
+update(tpf: float)
br.furb.inf.tcc.tankcoders
TankCoders
+main(String[] args)
+changeToInGameState()
+changeToMenuState()
br.furb.inf.tcc.util.game
GamePersistentProperties
+getInstance(): GamePersistentProperties
Lê informações
persistentes do arquivo
data/game-properties.cfg
br.furb.inf.tcc.tankcoders.menu
GameMenuGUI
+buildGUI(d: Display)
-display
JoinGameMenuGUI
+buildGUI(d: Display)
PrepareBattleMenuGUI
+buildGUI(d: Display)
-ready
MenuPixmapBackgrounds
+loadImages()
GameMenuState
+render(tpf: float)
+update(tpf: float)
-input
java.io
File
FileInputStream
br.furb.inf.tcc.util.ui
FengGuiUtils
+getRsrc(n: String): String
+getPixmapBackground(fp: String): PixmapBackground
MyTheme
+setUp(w: Window)
br.furb.inf.tcc.util.lang
GameLanguage
+getString(String key): String
+getInstance(): GameLanguage

Criação do cenário $\longrightarrow$ hiearquizando com **grafo de cena**.

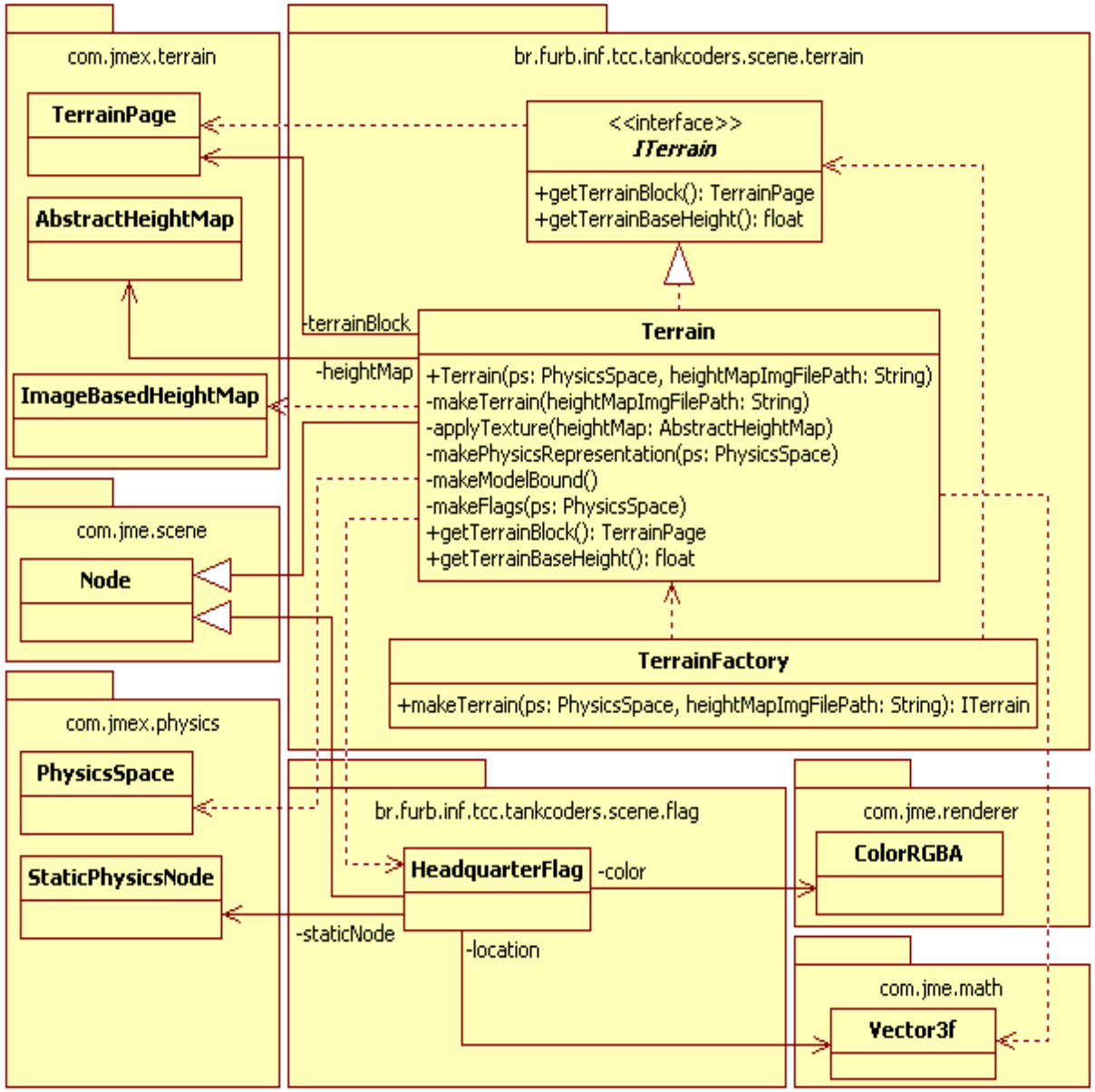
Elementos do cenário:

- `Sky`;
- `Terrain`;
- `Hud`;
- `Tanks`, podem ser:
  - ⋆ `TankM1Abrams`;
  - ⋆ `TankJadgePanther`.


RootNode
Sky
Terrain
Hud
TankM1Abrams
TankJadgePanther
SkyBox
TerrainPage
<<physicsNode>>
TerrainStaticNode
HQFlag1
HQFlag2
HQFlag3
HQFlag4
HudQuad
LabelTime
LabelTankName
LabelMessage
LabelTextWaitPlayers
LabelSpeed
LabelPrimaryGunBullets
LabelHealt
Sim
local
Não
<<physicsNode>>
TankChassiDyn
<<physicsNode>>
TankChassiStatic
LabelTankName
FrontSuspension
RearSuspension
SuspesionBase
LeftWheel
RightWheel
MainGun
<<physicsNode>>
WheelNodeDyn
Tyre
Sim
local
Não
<<physicsNode>>
MainGunDyn
<<physicsNode>>
MainGunStatic
MainGunSphere

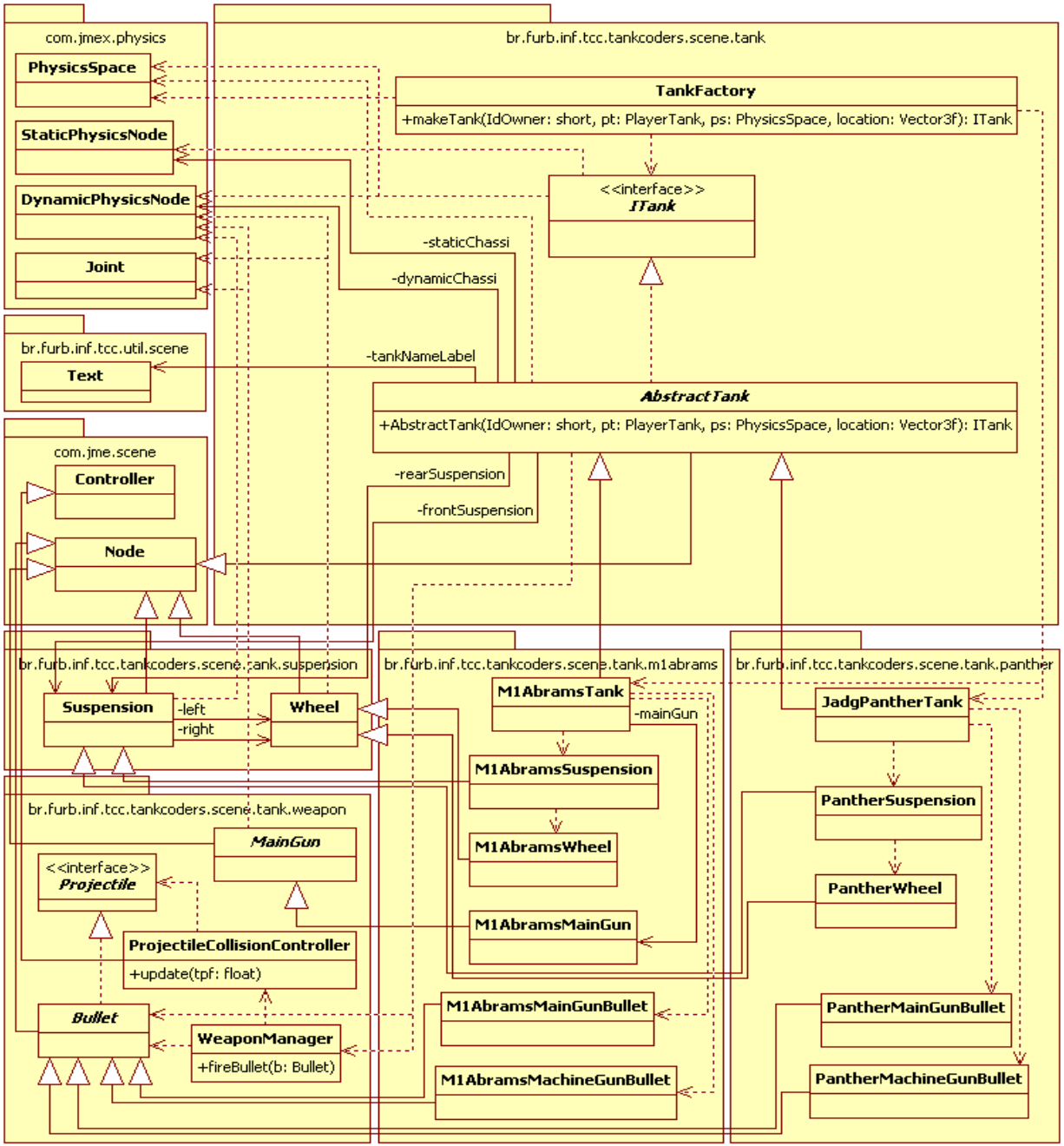
Terreno

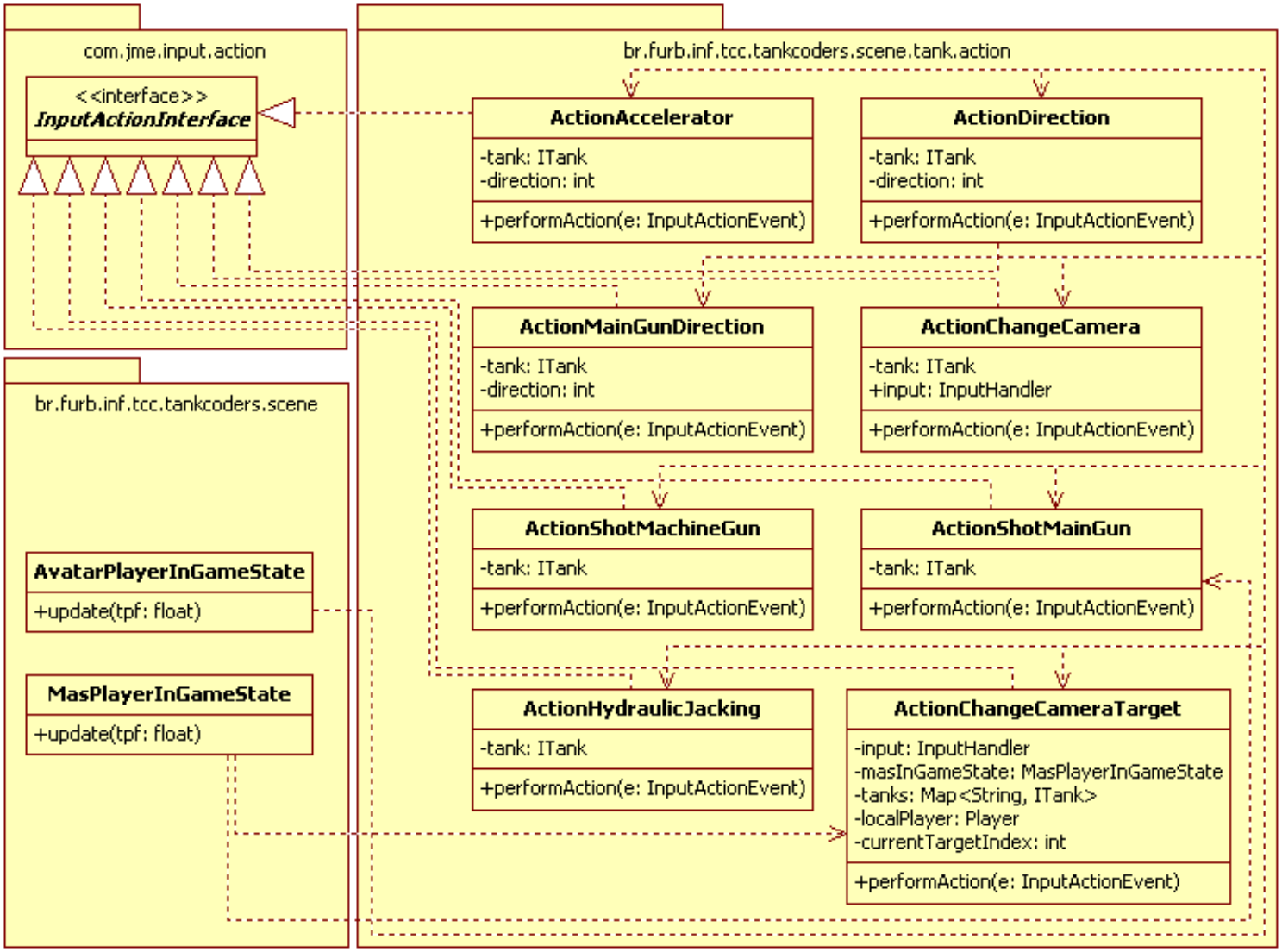
- Criação através da classe `TerrainFactory`
- Composto por uma malha de polígonos gerada a partir de um **mapa de alturas**
- Mistura de texturas para representar um terreno de areia
- Possui um **Quartel General (QG)** para cada time
- Delimita o QG com bandeiras (`HeadquarterFlag`)


com.jmex.terrain
br.furb.inf.tcc.tankcoders.scene.terrain
TerrainPage
<<interface>>
ITerrain
+getTerrainBlock(): TerrainPage
+getTerrainBaseHeight(): float
AbstractHeightMap
-terrainBlock
Terrain
-heightMap
ImageBasedHeightMap
+Terrain(ps: PhysicsSpace, heightMapImgFilePath: String)
-makeTerrain(heightMapImgFilePath: String)
-applyTexture(heightMap: AbstractHeightMap)
-makePhysicsRepresentation(ps: PhysicsSpace)
-makeModelBound()
-makeFlags(ps: PhysicsSpace)
+getTerrainBlock(): TerrainPage
+getTerrainBaseHeight(): float
com.jme.scene
Node
TerrainFactory
+makeTerrain(ps: PhysicsSpace, heightMapImgFilePath: String): ITerrain
com.jmex.physics
PhysicsSpace
br.furb.inf.tcc.tankcoders.scene.flag
com.jme.renderer
StaticPhysicsNode
HeadquarterFlag
-color
ColorRGBA
-staticNode
-location
com.jme.math
Vector3f

Tanques

- Criação através da classe `TankFactory`
- Pode ser integrado a um agente
- Pode ser controlado por um usuário remoto
- Pode ser controlado via teclado e mouse (*Actions*)
- Composição:
  - ⋆ Suspensões
    - ∗ Rodas
  - ⋆ Arma principal (Móvel ou Estática)
  - ⋆ Metralhadora (Fixa na base da arma principal)
  - ⋆ Chassi

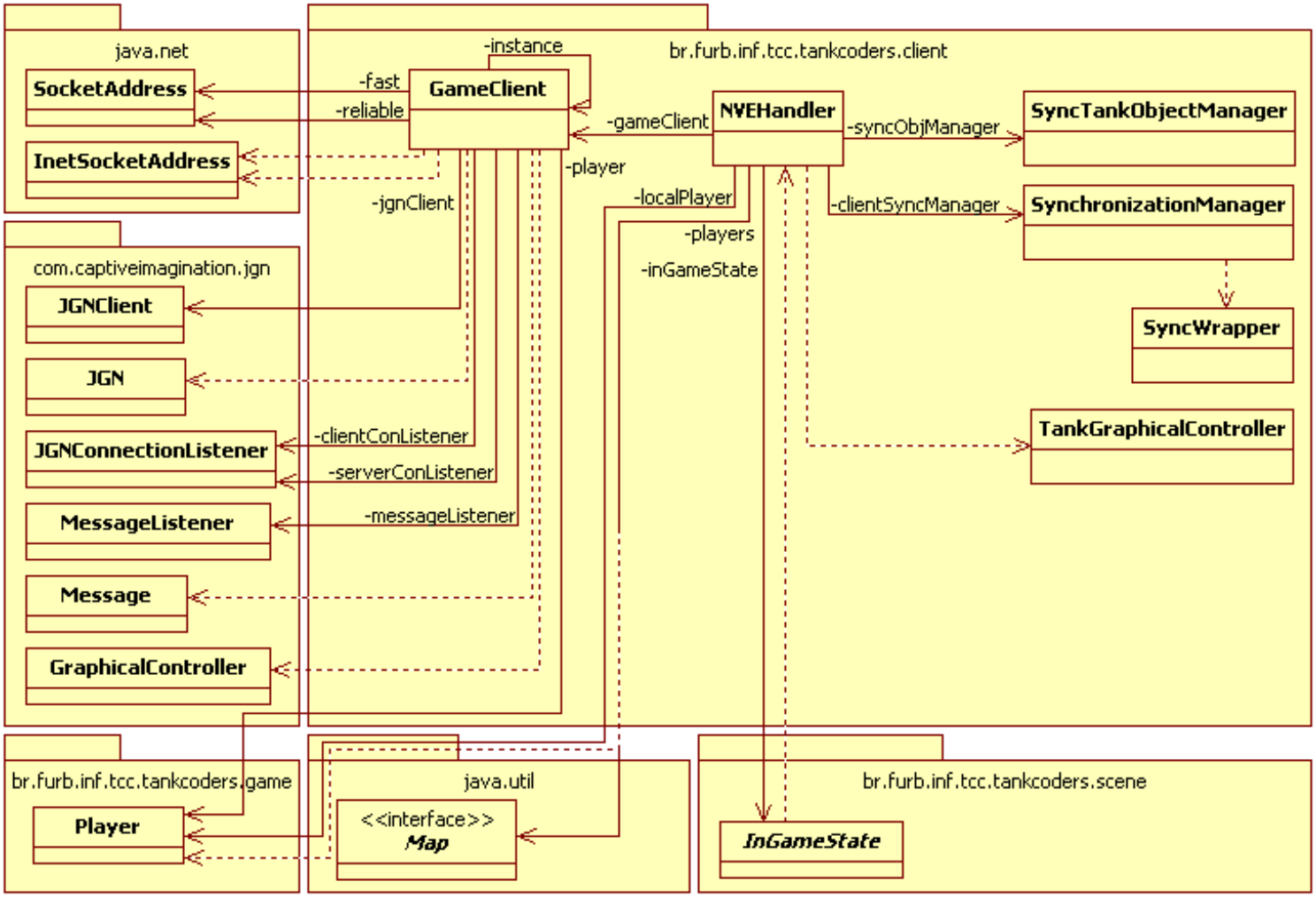

com.jmex.physics
PhysicsSpace
StaticPhysicsNode
DynamicPhysicsNode
Joint
br.furb.inf.tcc.util.scene
Text
com.jme.scene
Controller
Node
br.furb.inf.tcc.tankcoders.scene.tank
TankFactory
+makeTank(IdOwner: short, pt: PlayerTank, ps: PhysicsSpace, location: Vector3f): ITank
<<interface>>
ITank
-staticChassi
-dynamicChassi
-tankNameLabel
AbstractTank
+AbstractTank(IdOwner: short, pt: PlayerTank, ps: PhysicsSpace, location: Vector3f): ITank
-rearSuspension
-frontSuspension
br.furb.inf.tcc.tankcoders.scene.tank.suspension
Suspension
-left
-right
Wheel
br.furb.inf.tcc.tankcoders.scene.tank.m1abrams
M1AbramsTank
-mainGun
M1AbramsSuspension
M1AbramsWheel
M1AbramsMainGun
M1AbramsMainGunBullet
M1AbramsMachineGunBullet
br.furb.inf.tcc.tankcoders.scene.tank.panther
JadgPantherTank
PantherSuspension
PantherWheel
PantherMainGunBullet
PantherMachineGunBullet
br.furb.inf.tcc.tankcoders.scene.tank.weapon
MainGun
<<interface>>
Projectile
ProjectileCollisionController
+update(tpf: float)
Bullet
WeaponManager
+fireBullet(b: Bullet)


com.jme.input.action
<<interface>>
InputActionInterface
br.furb.inf.tcc.tankcoders.scene.tank.action
ActionAccelerator
-tank: ITank
-direction: int
+performAction(e: InputActionEvent)
ActionDirection
-tank: ITank
-direction: int
+performAction(e: InputActionEvent)
ActionMainGunDirection
-tank: ITank
-direction: int
+performAction(e: InputActionEvent)
ActionChangeCamera
-tank: ITank
+input: InputHandler
+performAction(e: InputActionEvent)
br.furb.inf.tcc.tankcoders.scene
ActionShotMachineGun
-tank: ITank
+performAction(e: InputActionEvent)
ActionShotMainGun
-tank: ITank
+performAction(e: InputActionEvent)
AvatarPlayerInGameState
+update(tpf: float)
MasPlayerInGameState
+update(tpf: float)
ActionHydraulicJacking
-tank: ITank
+performAction(e: InputActionEvent)
ActionChangeCameraTarget
-input: InputHandler
-masInGameState: MasPlayerInGameState
-tanks: Map<String, ITank>
-localPlayer: Player
-currentTargetIndex: int
+performAction(e: InputActionEvent)

Interação com o aplicativo servidor

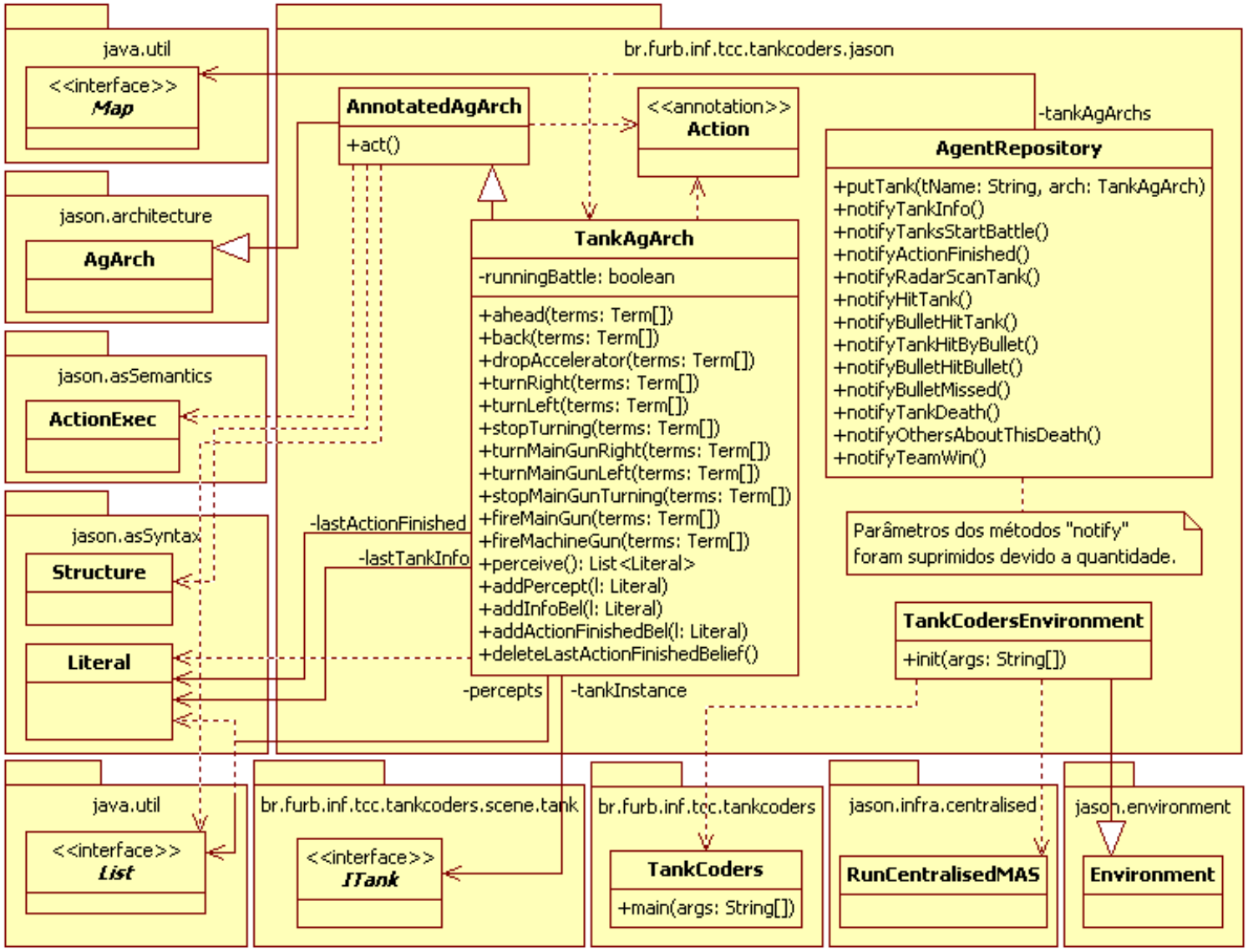
- Utilizando o *middleware* JGN
- Mensagens representadas por classes
- Todas as mensagens, exceto as de sincronização, derivam de RealtimeMessage (protocolo UDP)
- Mensagens classificadas de acordo com o *game state* em que ocorrem
- Uso de UDP *broadcasting* para mensagens comuns a todos os jogadores

CLASSE	CLASSIFICAÇÃO	SENTIDO
AbstractGameMessage	Ambos	Ambos
AllPlayersAreInGameState	Em jogo	Servidor -> Cliente
AnotherPlayerChangeModelResponse	Menu	Servidor -> Cliente
AnotherUserLogonRespose	Menu	Servidor -> Cliente
ChangeModel	Menu	Cliente -> Servidor
ChangeTeam	Menu	Cliente -> Servidor
InvalidChangeTeamResponse	Menu	Servidor -> Cliente
PlayerChangeTeamResponse	Menu	Servidor -> Cliente
PlayerInBattle	Em jogo	Cliente -> Servidor
PlayerLeftGame	Em jogo	Servidor -> Cliente
StartBattle	Menu	Servidor -> Cliente
Synchronize3DMessage	Em jogo	Cliente -> *broadcast*
SynchronizeArticulatedObject3DMessage	Em jogo	Cliente -> *broadcast*
SynchronizeCreateTankMessage	Em jogo	Cliente -> *broadcast*
TankActionShotMachineGun	Em jogo	Cliente -> *broadcast*
TankActionShotMainGun	Em jogo	Cliente -> *broadcast*
TankBulletHit	Em jogo	Cliente -> *broadcast*
TankDead	Em jogo	Cliente -> *broadcast*
TeamLeftGame	Em jogo	Cliente -> *broadcast*
UserLogoff	Menu	Cliente -> Servidor
UserLogon	Menu	Cliente -> Servidor
UserLogonFailedResponse	Menu	Servidor -> Cliente
UserLogonResponse	Menu	Servidor -> Cliente
UserReady	Menu	Cliente -> Servidor
UserUnready	Menu	Cliente -> Servidor


java.net
SocketAddress
InetSocketAddress
com.captiveimagination.jgn
JGNClient
JGN
JGNConnectionListener
MessageListener
Message
GraphicalController
br.furb.inf.tcc.tankcoders.client
-instance
GameClient
-fast
-reliable
-jgnClient
-player
-gameClient
NVEHandler
-syncObjManager
SyncTankObjectManager
-localPlayer
-players
-inGameState
-clientSyncManager
SynchronizationManager
SyncWrapper
TankGraphicalController
-clientConListener
-serverConListener
-messageListener
br.furb.inf.tcc.tankcoders.game
Player
java.util
<<interface>>
Map
br.furb.inf.tcc.tankcoders.scene
InGameState

Integração com o interpretador Jason

- Jason 1.1.1
- Componentes do Jason personalizados:
  - ⋆ `AgArch`
  - ⋆ `Environment`


java.util
<<interface>>
Map
br.furb.inf.tcc.tankcoders.jason
AnnotatedAgArch
+act()
<<annotation>>
Action
-tankAgArchs
AgentRepository
+putTank(tName: String, arch: TankAgArch)
+notifyTankInfo()
+notifyTanksStartBattle()
+notifyActionFinished()
+notifyRadarScanTank()
+notifyHitTank()
+notifyBulletHitTank()
+notifyTankHitByBullet()
+notifyBulletHitBullet()
+notifyBulletMissed()
+notifyTankDeath()
+notifyOthersAboutThisDeath()
+notifyTeamWin()
jason.architecture
AgArch
TankAgArch
-runningBattle: boolean
+ahead(terms: Term[])
+back(terms: Term[])
+dropAccelerator(terms: Term[])
+turnRight(terms: Term[])
+turnLeft(terms: Term[])
+stopTurning(terms: Term[])
+turnMainGunRight(terms: Term[])
+turnMainGunLeft(terms: Term[])
+stopMainGunTurning(terms: Term[])
+fireMainGun(terms: Term[])
+fireMachineGun(terms: Term[])
+perceive(): List<Literal>
+addPercept(l: Literal)
+addInfoBel(l: Literal)
+addActionFinishedBel(l: Literal)
+deleteLastActionFinishedBelief()
jason.asSemantics
ActionExec
jason.asSyntax
Structure
Literal
-lastActionFinished
-lastTankInfo
Parâmetros dos métodos "notify"
foram suprimidos devido a quantidade.
TankCodersEnvironment
+init(args: String[])
-percepts
-tankInstance
java.util
<<interface>>
List
br.furb.inf.tcc.tankcoders.scene.tank
<<interface>>
ITank
br.furb.inf.tcc.tankcoders
TankCoders
+main(args: String[])
jason.infra.centralised
RunCentralisedMAS
jason.environment
Environment

Implementação

Técnicas e Ferramentas Utilizadas

- IDE Eclipse 3.3 $\longrightarrow$ Java.
- Blender 2.43 $\longrightarrow$ Modelos dos Tanques 3D.
- Adobe Photoshop CS $\longrightarrow$ Texturas do terreno.
- Assembla Space $\longrightarrow$ Gerenciamento do projeto.
- Apache Ant $\longrightarrow$ Tarefas repetitivas.
- Padrões de Projeto: *Singleton*, *Command*, *Factory Method* e *Transfer Object*.

Operacionalidade da Implementação

Passos:

- Abrir aplicativo servidor, configurar e iniciar a batalha.
- Abrir um aplicativo cliente:
  - ⋆ Jogador **Avatar** ou **MAS**.
- Conectar-se ao servidor criado.
- Preparar os tanques.
- Definir-se como **Ready**.

Servidor notifica inicio da batalha $\longrightarrow$ clientes carregam cenário (estado AVD local)

Jogador Avatar

- Teclas:
  - ⋆ Movimentar Tanque: **W**, **S**, **A** e **D**.
  - ⋆ Atirar Arma Principal: **Barra Espaço**.
  - ⋆ Atirar Metralhadora: **Ctrl**.
  - ⋆ Movimentar Arma principal: **Direita** e **Esquerda**.
  - ⋆ Macaco hidráulico: **J**
  - ⋆ Modo Wireframed: **O**

Jogador MAS

- Teclas:
  - ⋆ Trocar foco da camera: **TAB**
  - ⋆ Modo Wireframed: **O**

Resultados e Discução

- *Middleware* JGN $\longrightarrow$ performance e abstração.
- *Engine* JME $\longrightarrow$ Grafo de cena $=$ performance, otimização e abstração.
- Cenário 3D com bom nível de imersão:
  - $\star$ Gravidade, massa, colisão, suspensões e disparo de projéteis.
- *Chase Camera* melhor visão para o jogador Avatar.
- Jason 1.1.1 $\longrightarrow$ desempenho nas ações.
- Abstração para o desenvolvedor AgentSpeak(L).
- **Orientação a Objetos** em aplicações gráficas.

Conclusão

Conclusão

- Simulador **TankCoders** e o estudo de POA.
- *Middlewares* e *frameworks* utilizados:
  - ⋆ JGN: pouca documentação e exemplos;
  - ⋆ Jason e plug-in Eclipse: Hübner, Bordini e Fronza;
  - ⋆ AgentSpeak: agentes cognitivos de forma declarativa e elegante;
  - ⋆ JME: Abstração e performance.

Extensões

- Desenvolver modelos 3D mais leves.
- Audio no ambiente.
- Efeitos de explosão.
- Novos modelos de terreno.
- Ações internas genéricas:
  - ⋆ *Pathfinding* por exemplo.
- Notificação de obstáculos do terreno aos agentes.
- Rodar sobre redes WAN (Internet).

Estatísticas

- Total de arquivos .java: 135
- Total de linhas: 16478
  - ⋆ Linhas de código: 81,19 %
  - ⋆ Comentários: 2,18 %

"O único lugar onde o sucesso vem antes do trabalho é no dicionário."

Albert Einstein