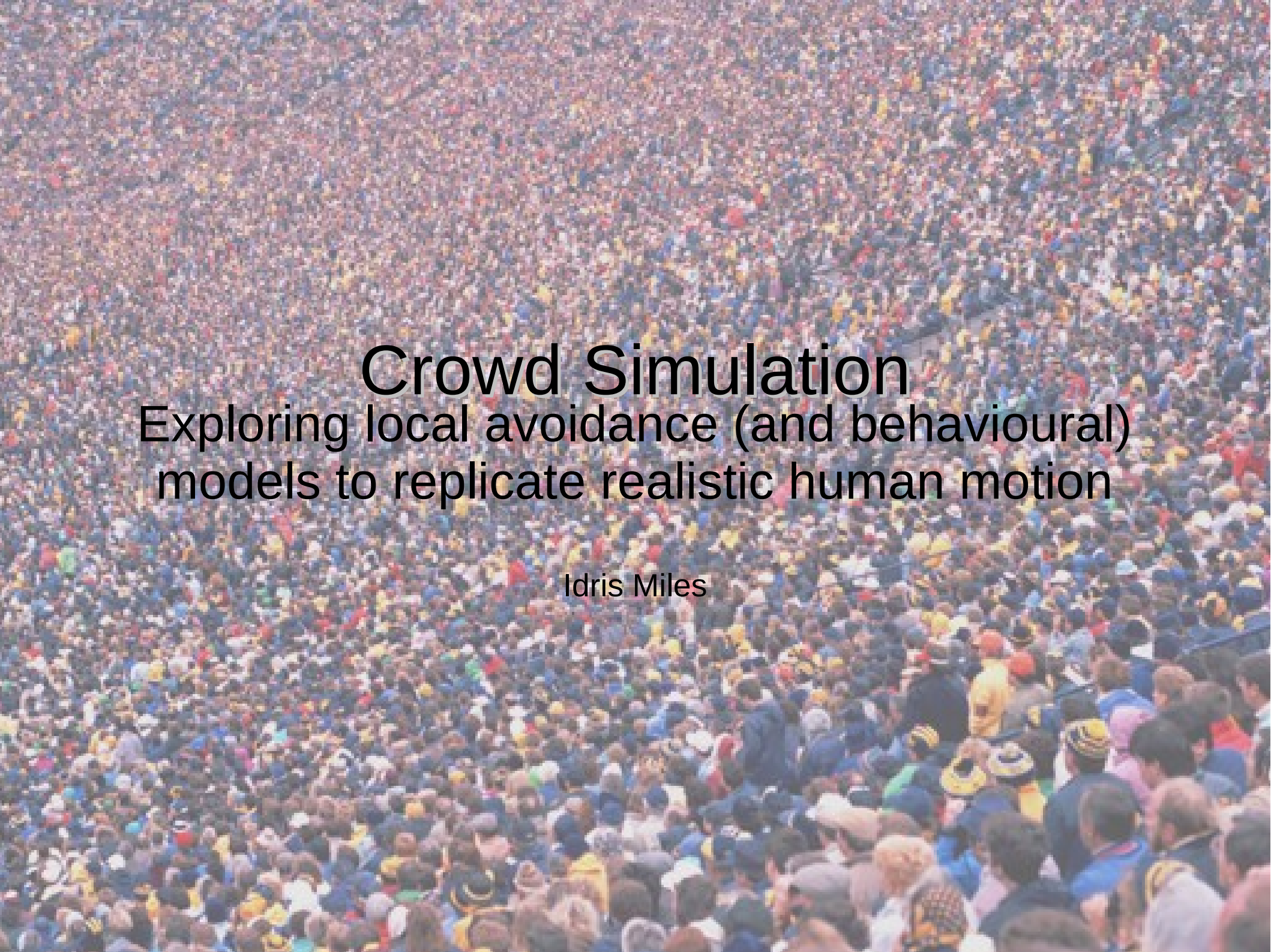


Crowd Simulation

Exploring local avoidance (and behavioural) models to replicate realistic human motion

Idris Miles

Main Focus

- Local avoidance models to replicate human motion:
  - Flocking rules
  - Social forces
  - **Reciprocal Velocity Obstacle (RVO) avoidance**
- Global navigation:
  - Flocking rules
  - A* path finding – way points
- Behaviour
  - Agent behaviour affecting paths chosen
  - Agent goal oriented behaviour – more AI direction

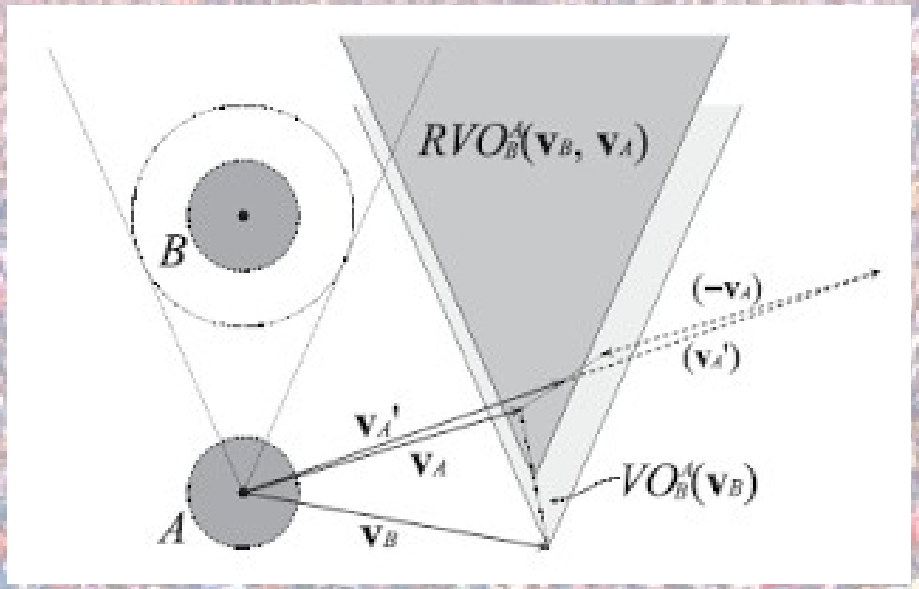
Local avoidance

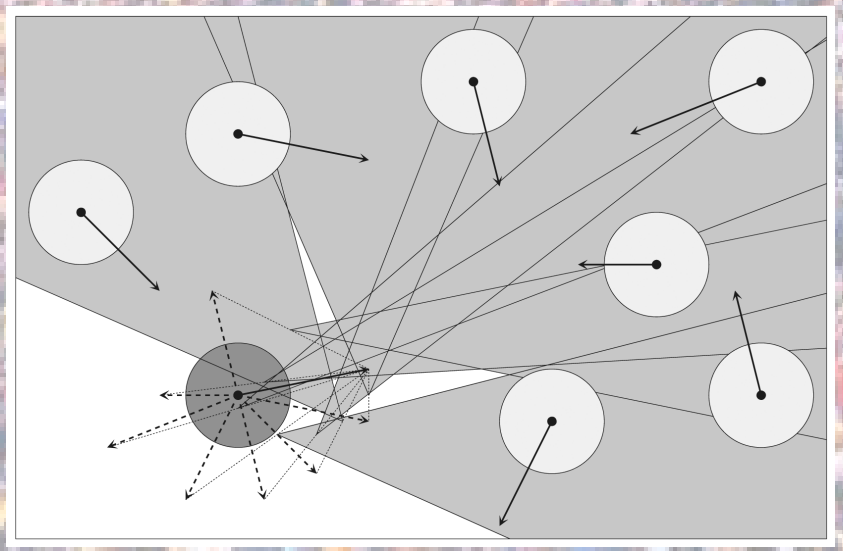
- Collision avoidance
- Smooth movement – perceiving obstacles in advance, avoid oscillation
- Neighbouring agents affect movement – Octree to speed up check
- Natural phenomena – forming groups, following in a line

Reciprocal **V**elocity **O**bstacle - RVO

- Based on Velocity Obstacle method:
  - $VO^{A}_{B}(v_B) = \{v_A \mid \lambda(p_A, v_A - v_B) \cap \mathbf{B} \oplus -\mathbf{A} = \emptyset\}$
- The velocity obstacle $VO^{A}_{B}(v_B)$ of agent **B** to **A** is the set of velocities V_A for **A** that will result in a collision with **B** moving at velocity V_B.
- Idea is to find a new velocity for the agent **A** that is outside the Velocity Obstacle of **B** (thus will not lead to a collision later on)
- Difference with RVO and VO – instead of finding velocity outside the VO of B we find the average between the velocity outside and the current velocity.
- Produces smoother movement

Reciprocal **V**elocity **O**bstacle - RVO





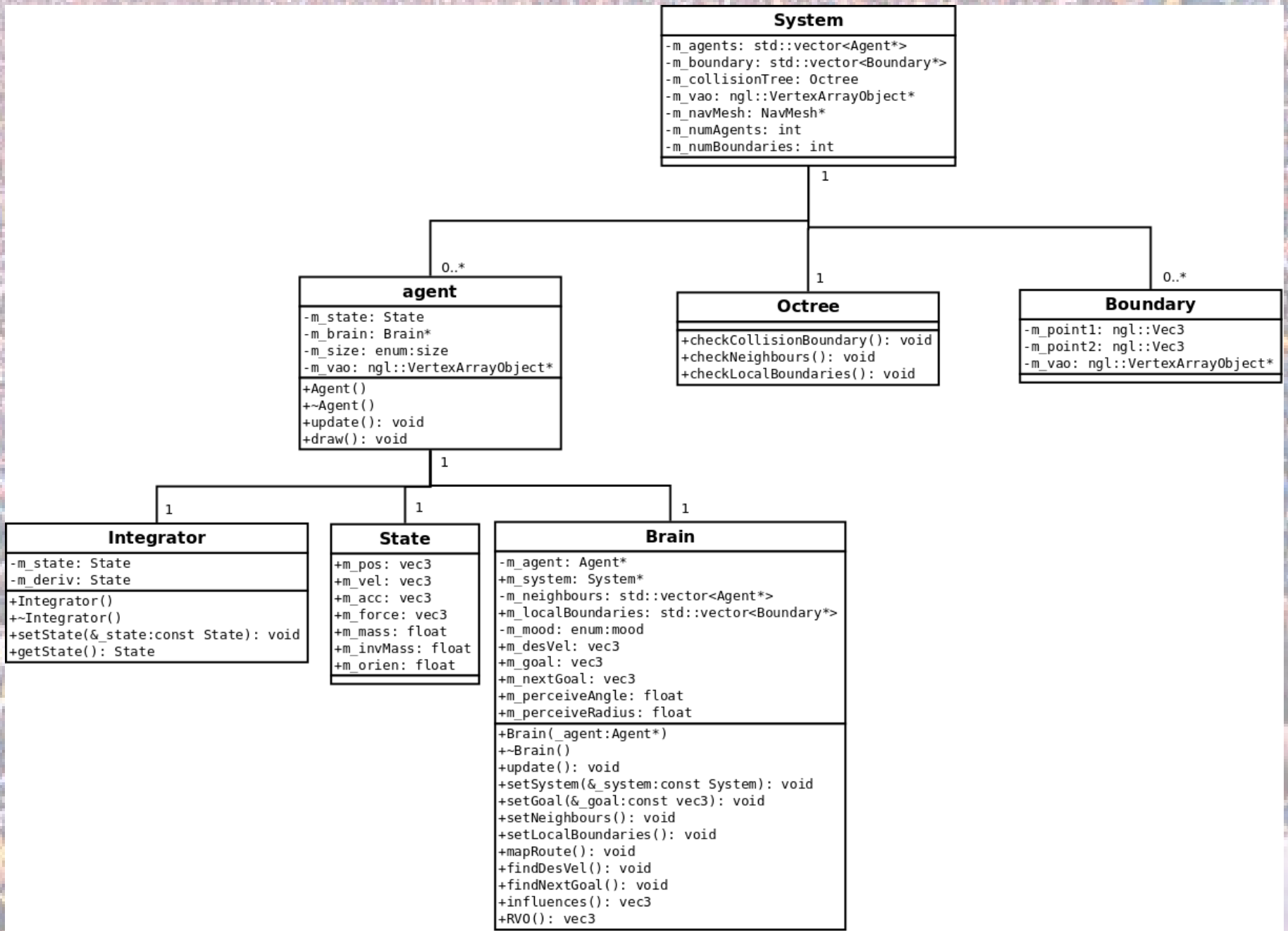
Behaviour

- Agent type:
  - Big, medium, small
- Agent brain:
  - Perceive & reacts to obstacles
  - Influences e.g. type of obstacle, big person = scared
  - Mood e.g. agitated – rush,
    relaxed – slow,
    happy – act normal,
    scared – quick sharp movement

Parallelize?

- The way RVO can be implemented, it may be possible to parallelize this method.
- Individual threads on the GPU would do the calculations for a single agent.
- Using GLSL shaders, pass certain attributes:
  - Neighbour positions & velocities,
  - Agents desired velocity & admissible velocities,
  - Agents current position and velocity,
- OR Compute shaders? OR CUDA/OpenCL?

UML design


System
-m_agents: std::vector<Agent*>
-m_boundary: std::vector<Boundary*>
-m_collisionTree: Octree
-m_vao: ngl::VertexArrayObject*
-m_navMesh: NavMesh*
-m_numAgents: int
-m_numBoundaries: int
1
0..*
1
0..*
agent
-m_state: State
-m_brain: Brain*
-m_size: enum:size
-m_vao: ngl::VertexArrayObject*
+Agent()
+~Agent()
+update(): void
+draw(): void
Octree
+checkCollisionBoundary(): void
+checkNeighbours(): void
+checkLocalBoundaries(): void
Boundary
-m_point1: ngl::Vec3
-m_point2: ngl::Vec3
-m_vao: ngl::VertexArrayObject*
1
1
1
1
Integrator
-m_state: State
-m_deriv: State
+Integrator()
+~Integrator()
+setState(&_state:const State): void
+getState(): State
State
+m_pos: vec3
+m_vel: vec3
+m_acc: vec3
+m_force: vec3
+m_mass: float
+m_invMass: float
+m_orien: float
Brain
-m_agent: Agent*
+m_system: System*
-m_neighbours: std::vector<Agent*>
+m_localBoundaries: std::vector<Boundary*>
-m_mood: enum:mood
+m_desVel: vec3
+m_goal: vec3
+m_nextGoal: vec3
+m_perceiveAngle: float
+m_perceiveRadius: float
+Brain(_agent:Agent*)
+~Brain()
+update(): void
+setSystem(&_system:const System): void
+setGoal(&_goal:const vec3): void
+setNeighbours(): void
+setLocalBoundaries(): void
+mapRoute(): void
+findDesVel(): void
+findNextGoal(): void
+influences(): vec3
+RVO(): vec3