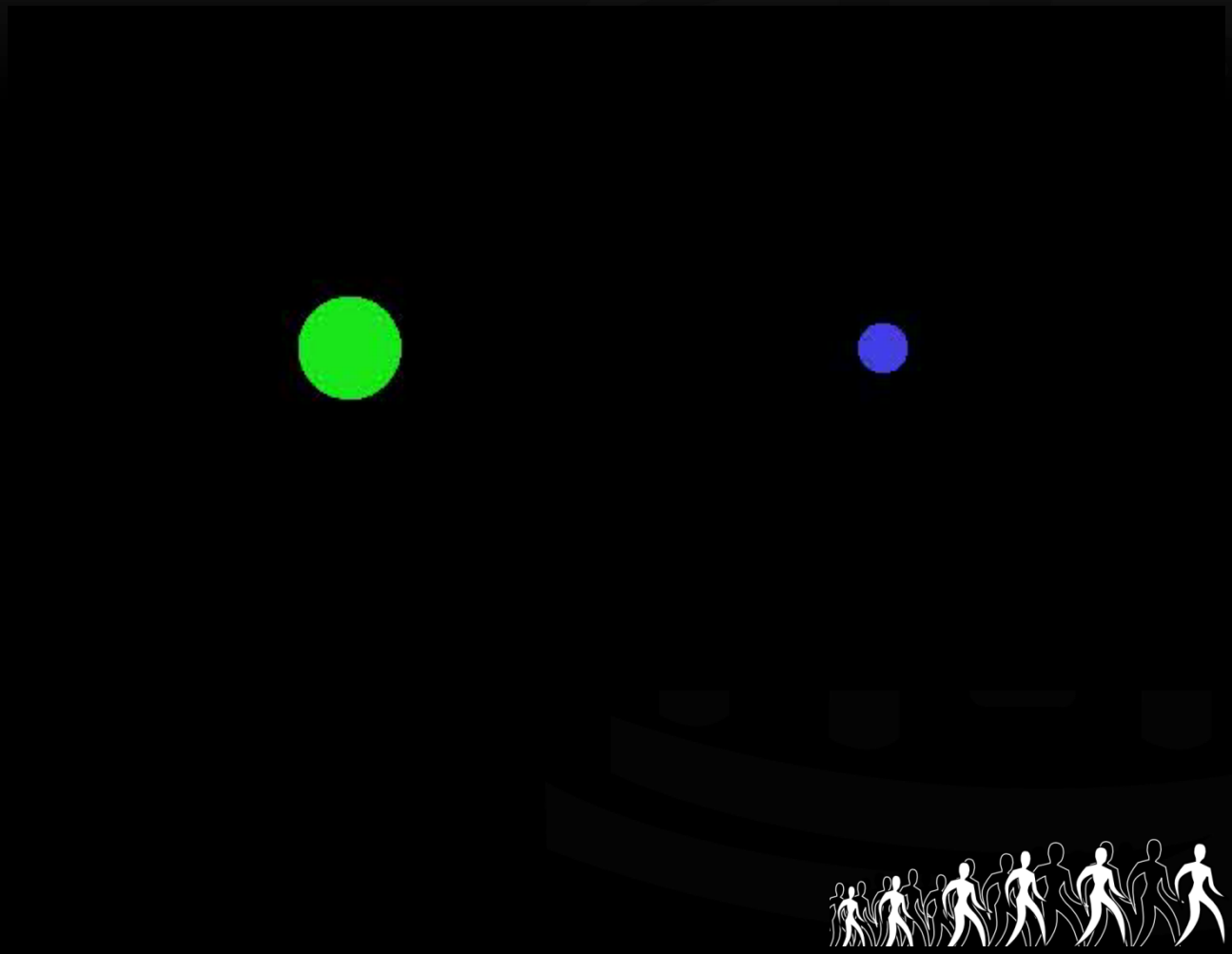
ENGINEERING
(MENGE, GIT, & OPENGL)

MENGE

- German for multitude, many, or crowd
- Modular pedestrian simulation framework
  - Black box pedestrian simulation
    - Simple experiments
    - Complex, time-varying scenarios
  - Development of novel pedestrian models
- Modular in two senses
  - Swap pedestrian models for comparisons
  - Construct complex scenarios from simple modules

MENGE



Menge

MENGE

Scene specification

```
<?xml version="1.0"?>
        <Experiment version="2.0">
        <Common time_step="0.1" visible_neighbors="0" />
        <Helbing agent_scale="2000" obstacle_scale="4000"
reaction_time="0.5" body_force="1200" friction="2400"
force_distance="0.015" />
        <Dummy stddev="0.05" />

        <AgentSet>
                <Common max_angle_vel="360" max_neighbors="10"
obstacleSet="1" neighbor_dist="5" r="0.19" class="1" pref_speed="1.04"
max_speed="2" max_accel="5" priority="0.0"/>
                <FD factor="1.57" buffer="0.9" />
                <Helbing mass="80" />
                <ORCA tau="3" tauObst="0.15" />

                <Agent p_x="2.0" p_y="0" />
        </AgentSet>
        …
```

Menge

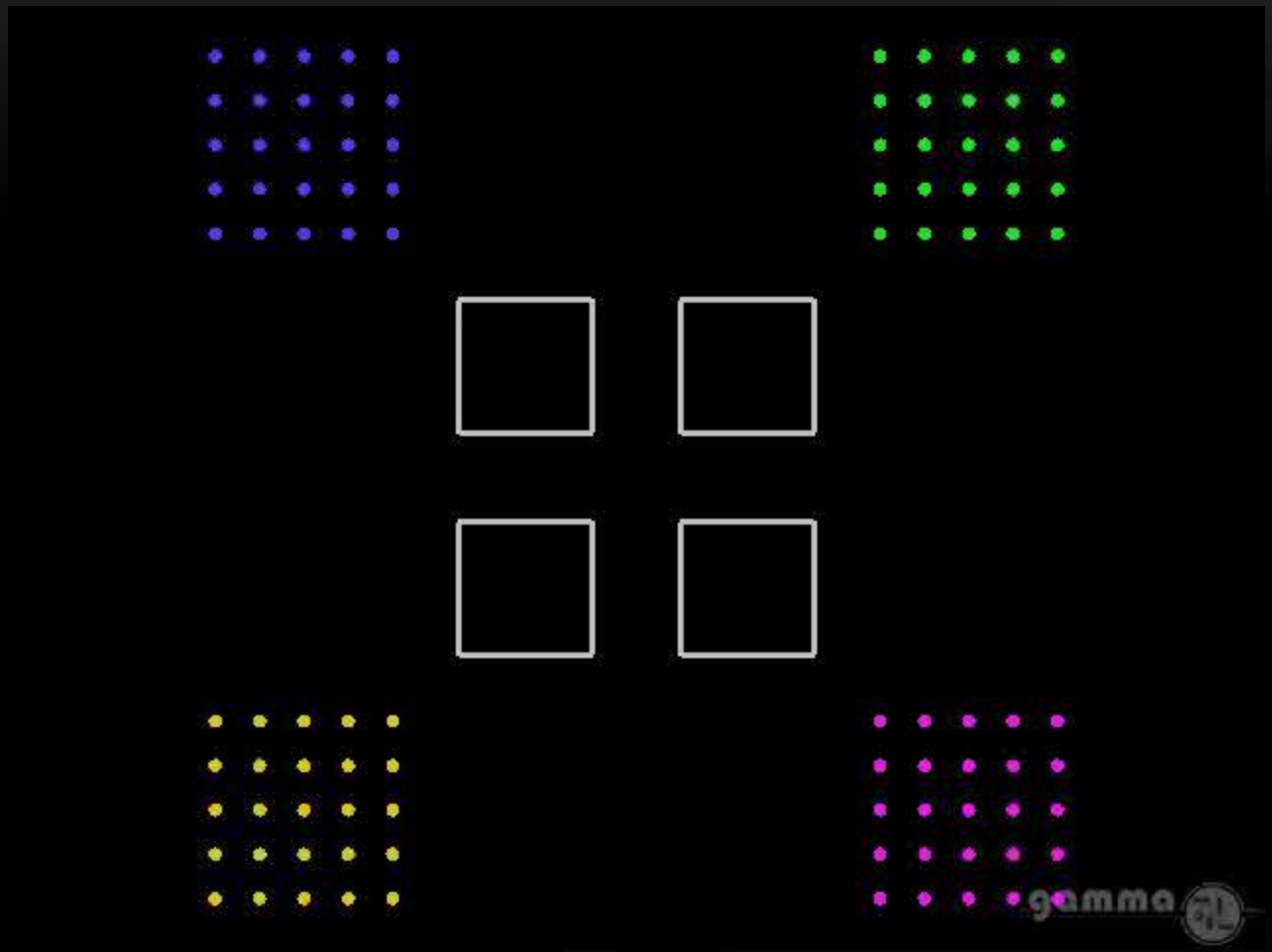
MENGE

Behavior specification

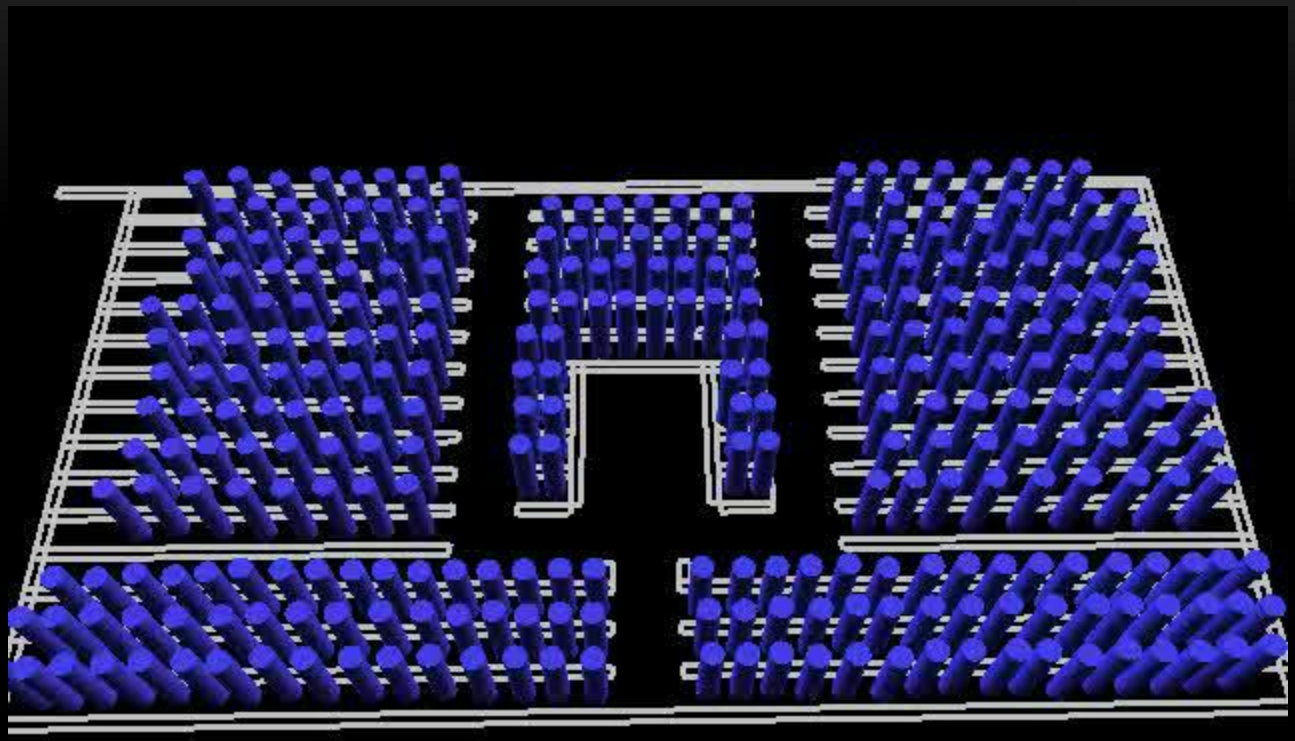
```
<?xml version="1.0"?><Population>
    <GoalSet id="0">
        <Goal type="point" id="0" x="-3" y="0.0"/>
        <Goal type="point" id="1" x="3" y="0.0"/>
    </GoalSet>
    <Behavior class="1" initVel="ZERO">
        <State name="Walk1" speedPolicy="min" final="0" >
            <AbsoluteGoal goalSet="0" goal="0" perAgent="0" />
            <VelComponent type="goal" weight="1.0" />
        </State>
        <State name="Reverse1" speedPolicy="min" final="0">
            <AbsoluteGoal goalSet="0" goal="1" perAgent="0" />
            <VelComponent type="goal" weight="1.0"/>
            <Action type="offset_property" property="r" dist="c"
            value="0.05" exit_reset="0"/>
            <Action type="scale_property" property="pref_speed" dist="c"
            value="1.05" exit_reset="0"/>
        </State>
        <Transition order="0" type="goal_circle" from="Walk1"
        to="Reverse1" radius="0.2" inside="1" />
        <Transition order="0" type="goal_circle" from="Reverse1"
        to="Walk1" radius="0.2" inside="1" />
    </Behavior>
    …
```

Menge

MENGE



MENGE



gamma

Menge

PHILOSOPHY

- Works with a particular kind of pedestrian model
  - Abstract function: f: (R^2 x S) → R^2
  - Given:
    - a preferred velocity in R^2
    - The local simulation state
  - Produces *actual* velocity

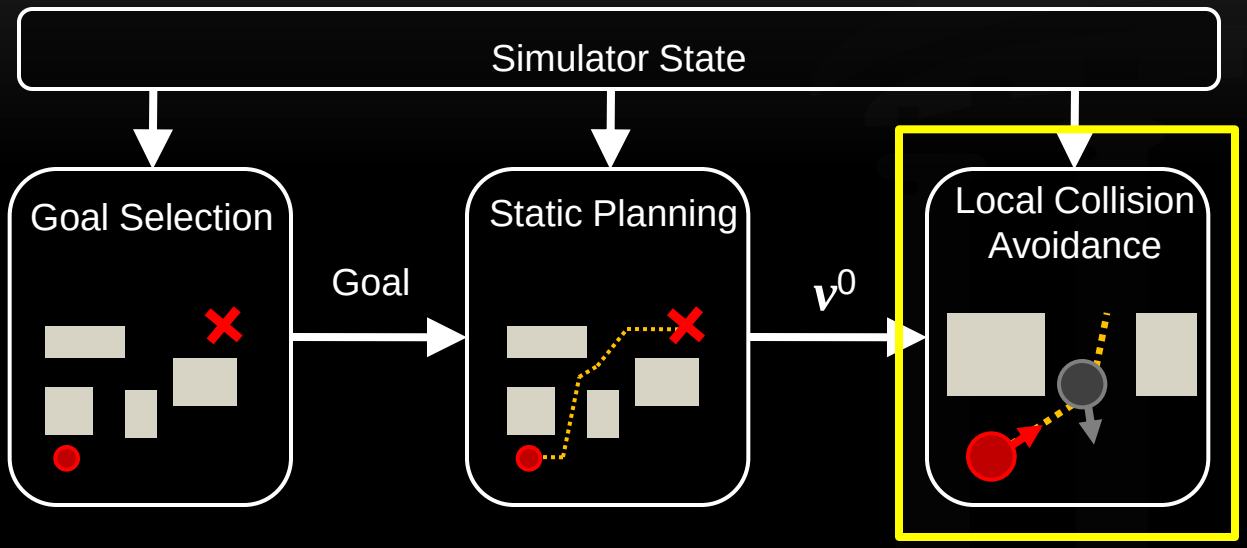
Menge

PHILOSOPHY

- This is a very common model
  - van den Berg et al., 2008, van den Berg et al., 2009, Helbing et al., 2000, Shukla et al., 2005, Karamouzas et al., 2009, Johansson et al., 2007, Narain et al., 2009, Zanlungo et al., 2010, Chraibi et al., 2010, Reynolds, 1995, cellular automata, and even more….
- (although there are some exceptions)
  - Treuille et al. 2006, Kapadia et al., 2011

Menge

PEDESTRIAN SIMULATOR ARCHITECTURE



- Simulation State: obstacles (static & dynamic), agents
- Goal Selection: High-order model of what the agent wants
- Static Planning: Plan to reach goal vs. *static* obstacles
- Local Collision Avoidance: Adapt plan because of other agents

Menge

ARCHITECTURE

- Factors out the computation of the inputs:
  - Preferred velocity
  - Local state
- Factors out the scene integration
- A pedestrian simply defines the function, f: (R^2 x S) → R^2
- Single scene and behavior specification can be used across arbitrary models.
  - Facilitates experimentation and comparison

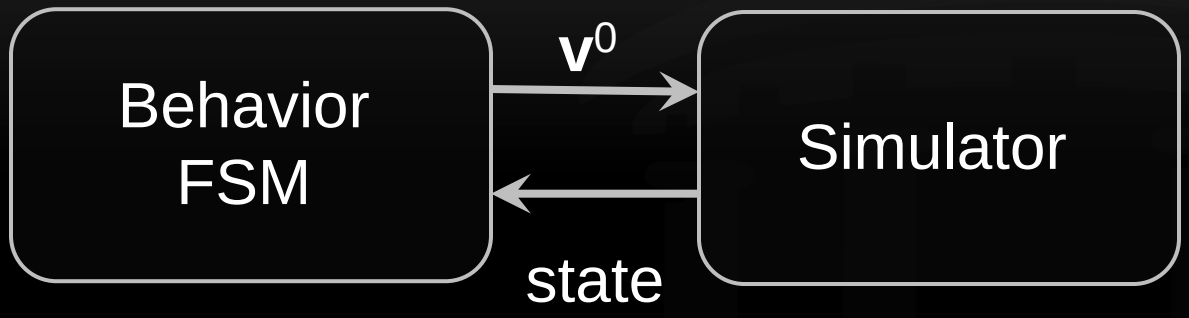
Menge

ARCHITECTURE

- Expandable
  - Uses a plug-in architecture to define new elements to be used at runtime
    - Pedestrian models
    - Spatial query structures
    - Elevation (for 2.5D simulation)
    - Path planning algorithms
    - Behavioral components

Menge

ARCHITECTURE

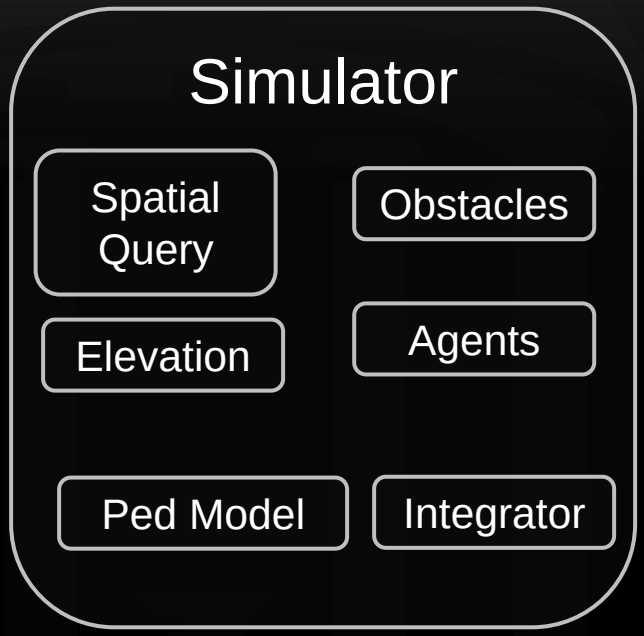


- Behavior → preferred velocity
  - Limited interpretation – motion in space
  - Where is the agent going, how is it getting there
  - Finite state machine → time-varying

Menge

SIMULATOR

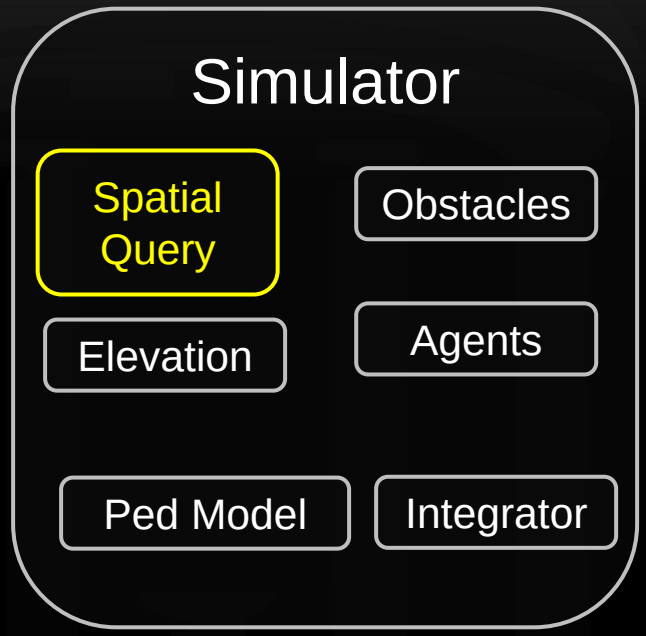
- Data
  - Maintains environment
    - Agent state
    - Static environment
    - Elevation data
- Functions
  - Nearest neighbor
  - Integrator
  - Pedestrian model



Menge

SIMULATOR

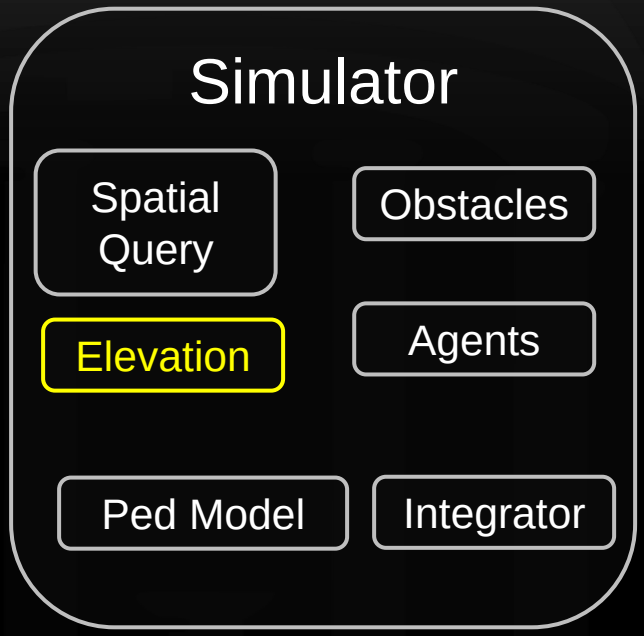
- Accelerated k-Nearest Neighbor Queries
  - Grid
  - KD-tree
  - Navigation mesh
  - Bounding volume hierarchy (BVH)
- Support k-nearest and visibility queries



Menge

SIMULATOR

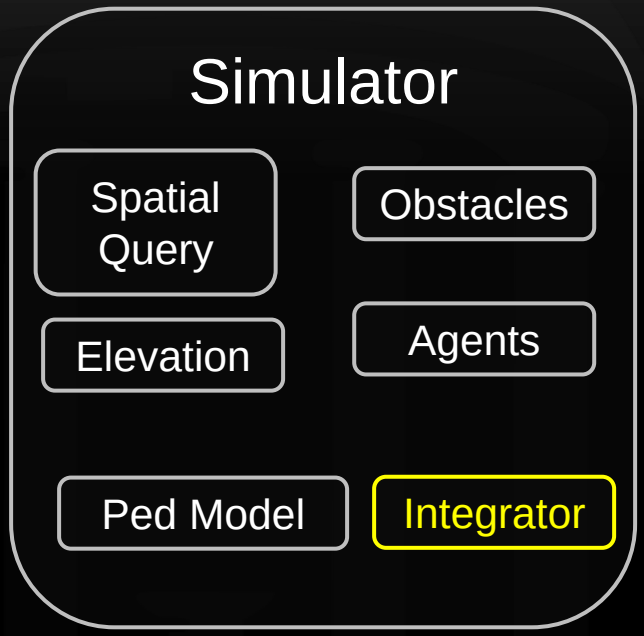
- Elevation queries
  - Height field
  - Function composition
  - Navigation mesh
- Support height at pos (x,y) and gradient as pos queries
  - Models 2.5D



Menge

SIMULATOR

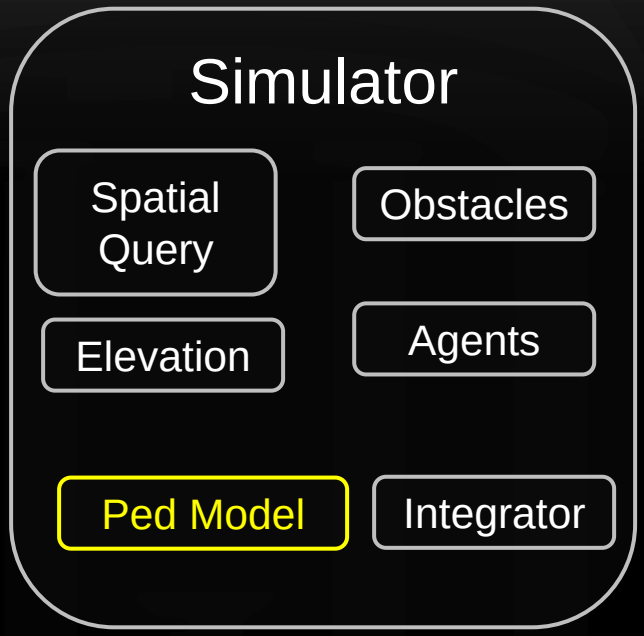
- Integrator (all explicit)
  - Euler-forward
  - RK-2
  - RK-4
- Not currently architected for user-configuration



Menge

SIMULATOR

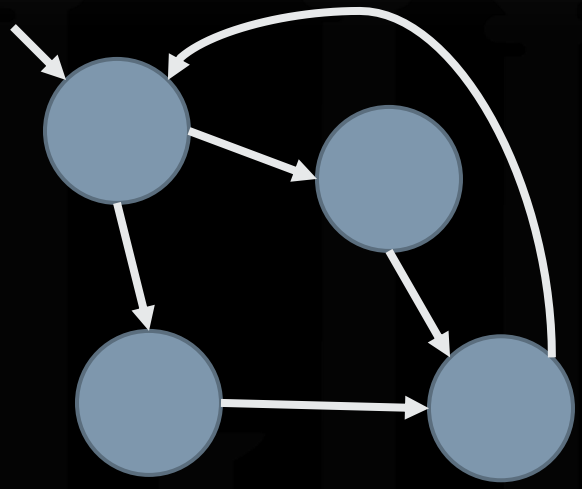
- Pedestrian model
  - “Dummy”
  - Helbing
  - ORCA
- Implement
  - newVelocity(pref, state)
  - XML parsing callbacks



Menge

BEHAVIORAL FSM

- States
  - Velocity components
  - Actions
- Transitions
- Goal
  - Specification and selection
- Specified by XML



Menge

STATES

- Primary purpose
  - Determine an agent's preferred velocity
  - `Velocity Component` (you'll be making one of these)
- Secondary purpose
  - Temporarily change the agent's state while in the state
  - `Action`

Menge

VELOCITY COMPONENT

- Model for computing preferred velocity
- Includes
  - Don’t move (zero velocity)
  - Constant velocity/direction
  - Walk straight to goal
  - Follow path from navigation mesh/roadmap
  - Guidance field

Menge

TRANSITIONS

- Conditions under which an agent changes state
  - Temporal
  - Spatial
  - Probabilistic
  - Agent state (including custom agent state)

Menge

GOALS

- Each agent has a goal *point*
- Which point?
  - The agent's position upon entering the state
  - Offset of agent's initial position
  - Mirror over world origin
  - Selecting point from goal specification

Menge

GOALS

- Goal Specification
  - `GoalSets`: a weighted collection of `Goals`
    - `Goal` selected from set
      - Explicitly
      - Randomly (with uniform or weighted probability)
      - Euclidian distance (near/far)
      - Path distance (nav mesh/road map)

Menge

GOALS

- `Goal`
  - Point
  - Disk
  - Axis-aligned box
  - Oriented box

Menge

MENGE

Behavior specification

```
<?xml version="1.0"?><Population>
    <GoalSet id="0">
        <Goal type="point" id="0" x="-3" y="0.0"/>
        <Goal type="point" id="1" x="3" y="0.0"/>
    </GoalSet>
    <Behavior class="1" initVel="ZERO">
        <State name="Walk1" speedPolicy="min" final="0" >
            <AbsoluteGoal goalSet="0" goal="0" perAgent="0" />
            <VelComponent type="goal" weight="1.0" />
        </State>
        <State name="Reverse1" speedPolicy="min" final="0">
            <AbsoluteGoal goalSet="0" goal="1" perAgent="0" />
            <VelComponent type="goal" weight="1.0"/>
            <Action type="offset_property" property="r" dist="c"
            value="0.05" exit_reset="0"/>
            <Action type="scale_property" property="pref_speed" dist="c"
            value="1.05" exit_reset="0"/>
        </State>
        <Transition order="0" type="goal_circle" from="Walk1"
        to="Reverse1" radius="0.2" inside="1" />
        <Transition order="0" type="goal_circle" from="Reverse1"
        to="Walk1" radius="0.2" inside="1" />
    </Behavior>
    …
```

MENGE

- Code available on sourceforge
  - git clone git://git.code.sf.net/p/menge/code menge-code
- Documentation at
  - http://gamma.cs.unc.edu/PBlock/
  - More will be coming today
- Hint of homework…

Menge

GIT

- Source control
  - http://git-scm.com/
  - Like SVN, CVS, and other but with some philosophical differences
  -

Menge

GIT

- Three reasons you want to use it
  - You have a full copy of the repository on your machine
  - Files are not versioned separately – changes are lumped together
  - That's how you get the code for your homework (and how you'll inherit the inevitable bug fixes)

Menge

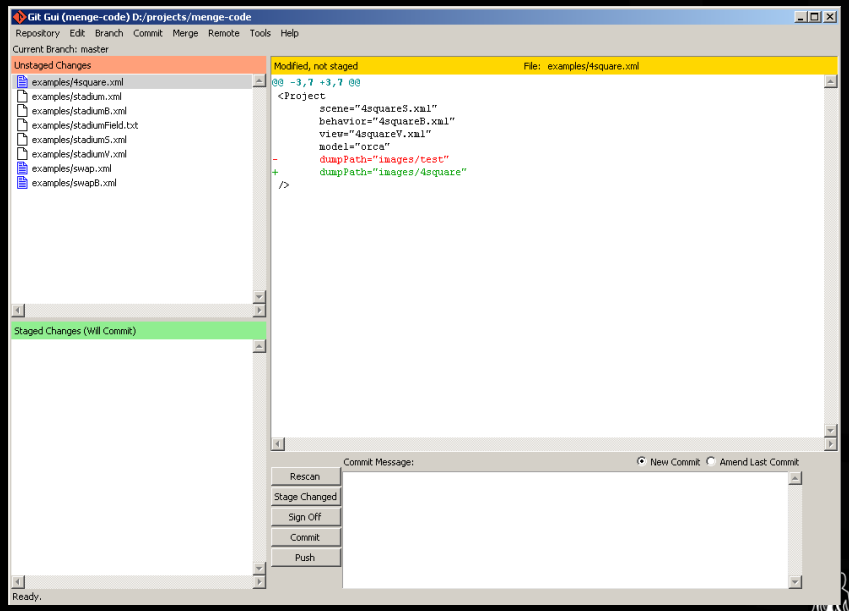
GIT

- 90% of what you need to do you can do with 2 commands
  - git add (adding changes to be committed)
  - git commit (performing the commit)
- The rest of what you need
  - git clone (copying a repository from existing repository)
  - git branch (creating a branch)
  - git merge (merge one branch into another)

Menge

GIT

- 90% of what you need to do you can do with 2 commands
  - git add (adding changes to be committed)
  - git commit (performing the commit)



Menge

GIT

- The rest of what you need
  - git clone (copying a repository from remote repository)
  - git branch (creating a branch)
  - git merge (merge one branch into another)
  - git pull (update branch from remote repository)
  - git stash (temporarily stash your changes)

Menge

OPENGL

- State-based drawing
  - A “context” exists with certain state
    - Drawing color, line width, lighting information, transformation matrix, etc.
  - We will use “immediate” mode
    - Simply perform a sequence of calls consisting of state changes and drawing commands

Menge

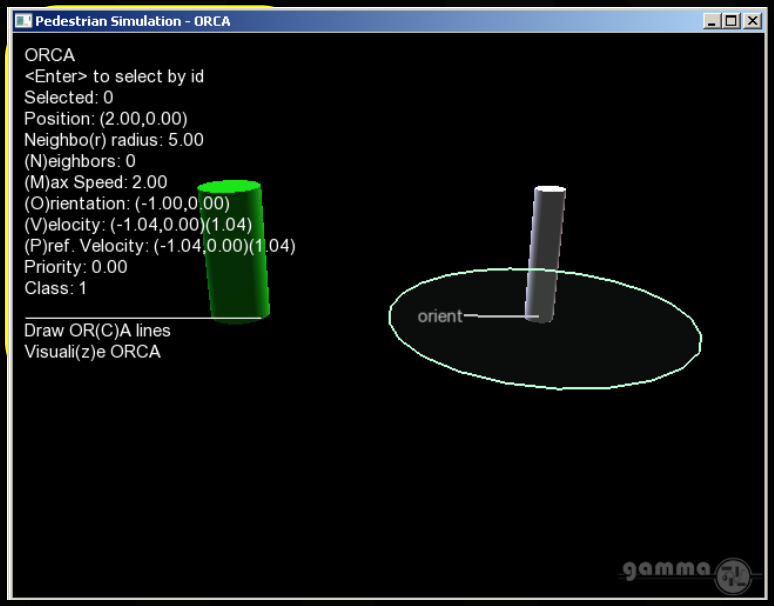
OPENGL

- Menge has a class, SG::Context, which simplifies much of what you want to do
  - Handles drawing text to the screen
  - Being able to select agents
- This is the limit of what you need to do
  - See SimRuntime/BaseAgentContext.h

Menge

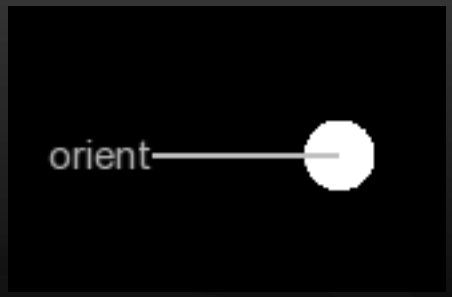
SG::CONTEXT

- Has several important functions
  - drawUIGL
    - Draw things in screen space (like an overlay)
  - draw3DGL
    - Draw things in the actual 3D world



Menge

OPENGL



- Example: drawing orientation

```
// Acquire orientation
Vector2 orient = agt->_orient;
// Determine end point of vector
Vector2 endPoint = agt->_pos + orient;
// Disable alpha blending
glDisable( GL_BLEND );
// Set the drawing color
glColor3f( 0.75f, 0.75f, 0.75f );
// Inform the system we're going to be drawing lines
glBegin( GL_LINES );
// A sequence of vertex - each 2 are considered one line
glVertex3f( agt->_pos.x(), Y, agt->_pos.y() );
glVertex3f(endPoint.x(), Y, endPoint.y() );
// Stop drawing
glEnd();
// draw Text
writeTextRadially( "orient", endPoint, agt->_orient );
```

Menge

OPENGL

- Changing state – and restoring it
  - Because of the “state”, what gets drawn depends on the state.
    - Rule 1: If you change the state, put it back
  - OpenGL gives a good way of doing this
    - glPushAttrib(…)
      - Push some aspects of the current state onto a stack (the ones you intend to change)
    - glPopAttrib()
      - Restore the last state attributes back

Menge

OPENGL

- Changing state – and restoring it
  - Example
    - glColor3f(1.f, 0.f, 0.f); // red
    - glPushAttrib(GL_COLOR_BUFFER_BIT)
    - glColor3f(1.f, 1.f, 1.f); // white
    - glPopAttrib();
    - // draw lines – what color are the lines

Menge

OPENGL

- Changing state – and restoring it
  - Always, always, always, make sure that you match up your glPushAttrib and glPopAttrib calls.
    - If you fail to do this, you could destroy the stack and crash your program

Menge

OPENGL

- Drawing primitives
  - glBegin(…) and glEnd()
  - How OpenGL interprets vertices

```
glBegin( PRIMITIVE_TYPE )
glVertex3f( x0, y0, z0 )
glVertex3f( x1, y1, z1 )
glVertex3f( x2, y2, z2 )
glVertex3f( x3, y3, z3 )
glEnd()
```

Menge

OPENGL

- Drawing primitives
  - Options for glBegin():
    - GL_POINTS – every vertex a point (see glPointSize(size))
    - GL_LINES – every two vertices a line segment (see glLineWidth(w))
    - GL_LINE_STRIP – draw a poly line – n vertices → n-1 connected segments
    - GL_LINE_LOOP – draw a closed poly line – n verts → n connected segments
    - GL_TRIANGLES – every three vertices is a triangle
    - GL_TRIANGLE_FAN – vertices define a fan of triangles
    - GL_TRIANGLE_STRIP – vertices define a strip of triangles
    - GL_QUADS – 4 verts defines a quad
    - GL_QUAD_STRIP – vertices define a strip of quads
    - GL_POLYGON – vertices define a ***convex*** polygon

Menge

OPENGL

- Drawing controlling how things appear
  - Color
    - `glColor3f( r, g, b)` or `glColor4f( r, g, b, a )`
  - Transparency

```
glPushAttrib( GL_ENABLE_BIT )
glEnable( GL_BLEND )
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

    - Just memorize these

Menge

OPENGL

- Drawing text
  - BaseAgentContext has several helper functions for drawing into the 3D world – currently this assumes y = 0.
    - writeText(text, 2D position)
    - writeTextRadially(text, 2D position, 2D direction)
    - writeAlignedText(text, 2D position, alignment)
  - All are shortcuts to the SG::TextWriter class

Menge

OPENGL

- What's in front?
  - OpenGL uses a "depth" buffer.
  - Two parts:
    - Store the depth of the closest thing at each pixel
    - Test against the closest pixel (if behind, don't draw)
  - There are times when you want to disable one or the other of these

Menge

OPENGL

- What if I want something ALWAYS behind?
  - Disable writing to depth buffer
    - glDepthMask(GL_FALSE)
  - Draw it first
  - Enable writing to depth buffer
    - glDepthMask(GL_TRUE)
  - Draw everything else

Menge

OPENGL

- What if I want something ALWAYS in front?
  - Turn off the depth test
    - glDisable(GL_DEPTH_TEST)
  - Draw it
  - Enable depth test
    - glEnable(GL_DEPTH_TEST)

Menge

OPENGL

- Use the glPushAttrib/glPopAttrib to put things back
  - glPushAttrib(GL_DEPTH_BUFFER_BIT)
  - glDepthMask(GL_FALSE)
  - // draw
  - glPopAttrib()

Menge

OPENGL

- Use the glPushAttrib/glPopAttrib to put things back
  - glPushAttrib(GL_ENABLE_BIT)
  - glDisable(GL_DEPTH_TEST)
  - // draw
  - glPopAttrib()
- In both cases, I don’t know what the state was *before* I needed to change the state, but I’ve put it back

Menge

QUESTIONS?