



Computational Representation of Stories in Games

Arnav Jhala
27/02/09

Game Development
Spring 2009

Storytelling Concepts (Recap)

- Environment/Setting
- Events
- Characters
- Goals
- Causality
- Drama
- Emotional Impact

Storytelling in Games (Recap)

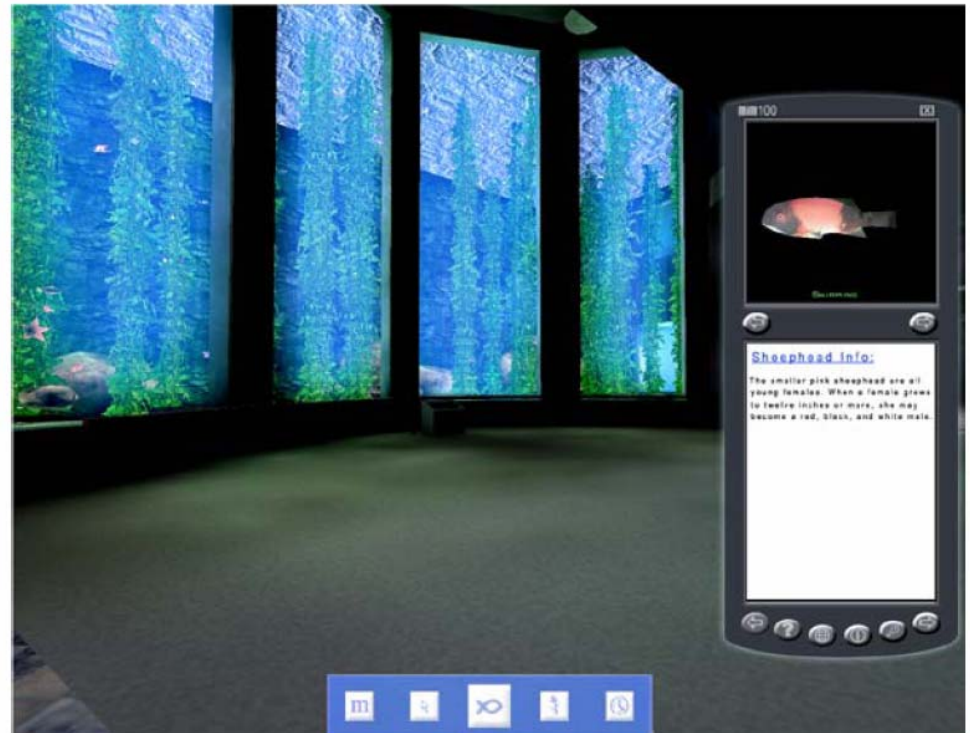
- Problem with agency
 - Rewards (player actions -> outcome)
 - Coherence of narrative
 - Dramatic and Emotional impact
- From linear stories to branching storylines
- Two types of narratives
 - Character-centric
 - Plot-centric

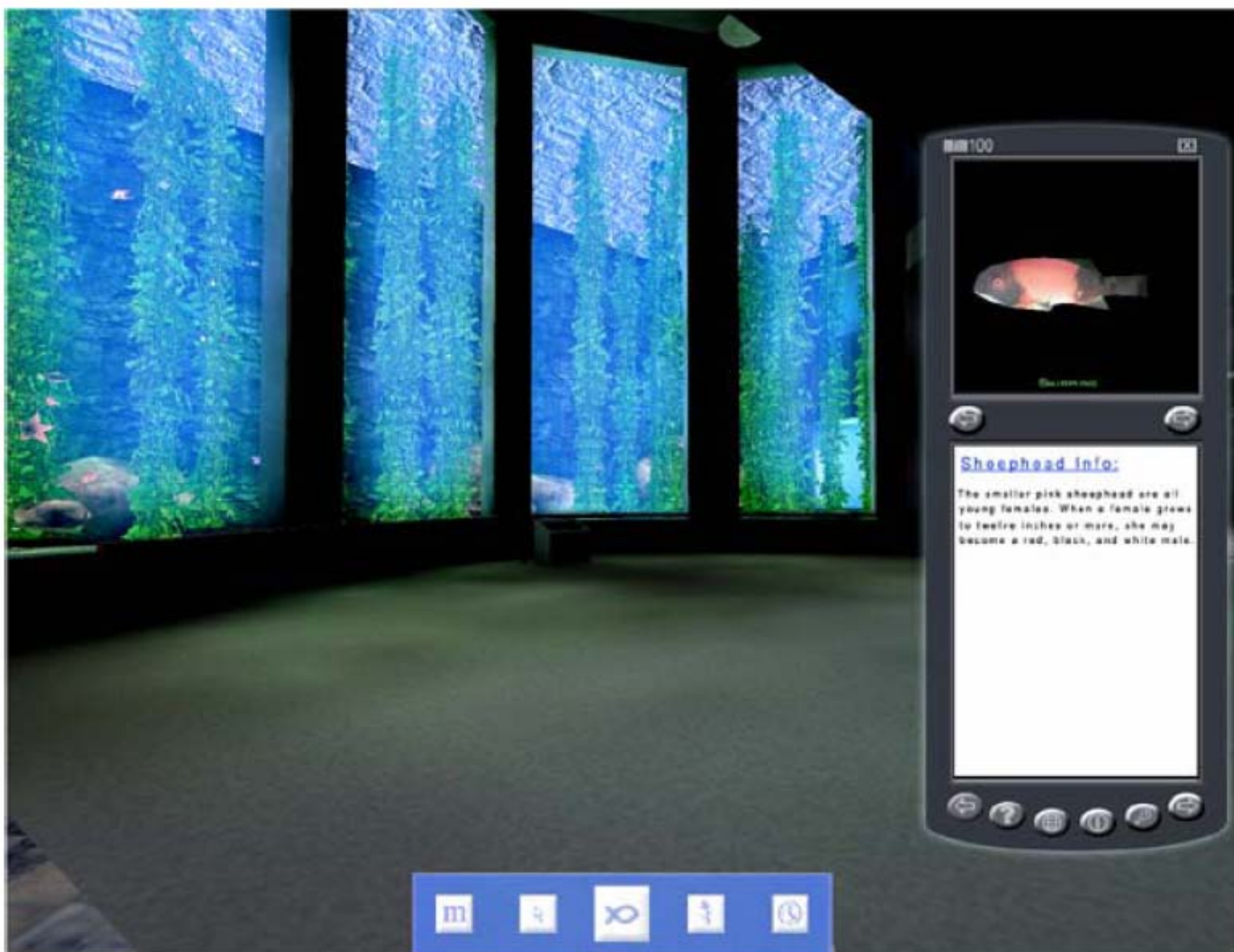
Examples



Façade

Mimesis





000100

[X]



Sheephead

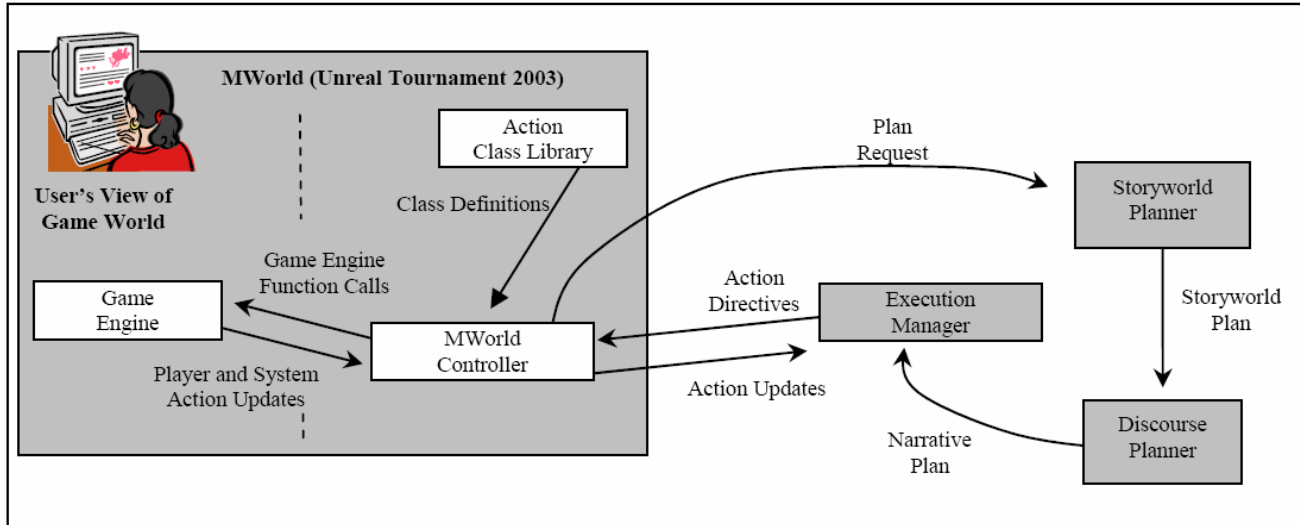
Sheephead Info:

The smaller pink sheephead are all young females. When a female grows to twelve inches or more, she may become a red, black, and white male.



Mimesis Architecture

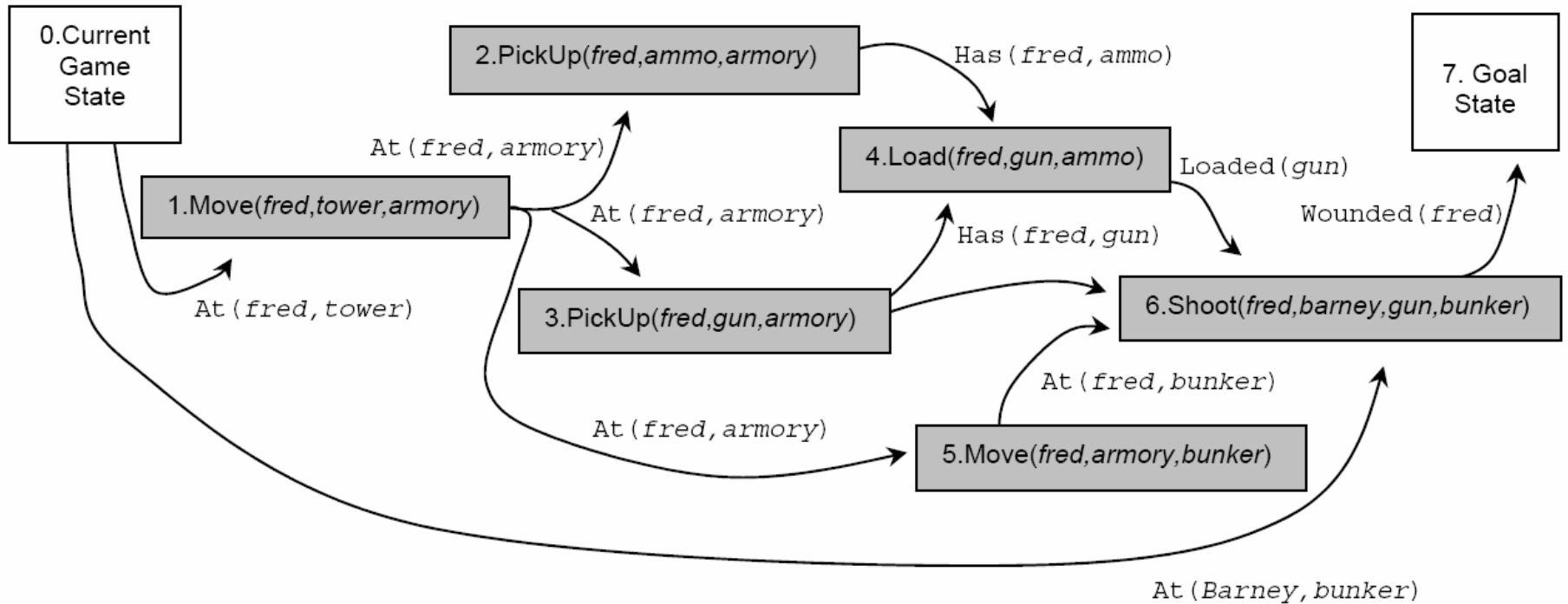
- Pre-planned story
 - Goal Oriented
 - Causality is represented
- Interactivity
 - Accomodation and Intervention



Story Elements

- Game State
- Actions/Events
 - Preconditions
 - Effects
- Goals
 - Overall story goals
 - Character sub-goals

Story Representation



Action Representation

Operator Shoot (*?shooter ?target ?weapon ?room*)

Constraints:

(health-level *?target ?t_health*)

(damage-level *?weapon ?damage_amount*)

Preconditions:

(has-weapon *?shooter ?weapon*)

(has-ammo *?weapon*)

(in-room *?shooter ?room*)

(in-room *?target ?room*)

Effects:

(health-level *?target*


(- *?t_health ?damage_amount*))

Unreal Implementation


```
class Shoot extends Action;
```

```
var MController Agent;  
var Pawn MyTarget;  
var Weapon MyWeapon  
var int OriginalHealth;  
var int precondResult;  
var int effectsResult;
```

```
function int CheckPreconds(){  
    if_(Agent.Pawn.Weapon != MyWeapon)  
        {return 0;}  
    else if (!(Agent.Pawn.Weapon.HasAmmo()))  
        {return 1;}  
    else if (Agent.Room != MyTarget.Room)  
        {return 2;}  
    else  
        {return -1;}  
}
```



CheckPreconds() checks each precondition from the corresponding operator in the order in which they are defined there. When a precondition does not hold in the current game state, an integer identifying the failed precondition is returned. If all preconditions hold, the function returns -1.



CheckEffects() runs after the body of the action completes. It verifies each of the operator's effects. When an effect does not hold in the current game state, an integer identifying the failed effect is returned. If all effects hold, the function returns -1.

Unreal Implementation (Contd.)

```
function int CheckEffects(){
    if( (MyTarget.Health >= OriginalHealth))
        {return 0;}
    else {return -1;}
```

```
function void Body(){
    local int fireMode;
    OriginalHealth = MyTarget.Health;
    Agent.Pawn.SetPhysics(PHYS_None);
    Agent.Pawn.SetViewRotation(rotator(MyTarget.Location -
        Agent.Pawn.Location));
    fireMode = Agent.Pawn.Weapon.BestMode();
    Agent.Pawn.Weapon.StartFire(fireMode);
}
```

state Executing {

Begin:

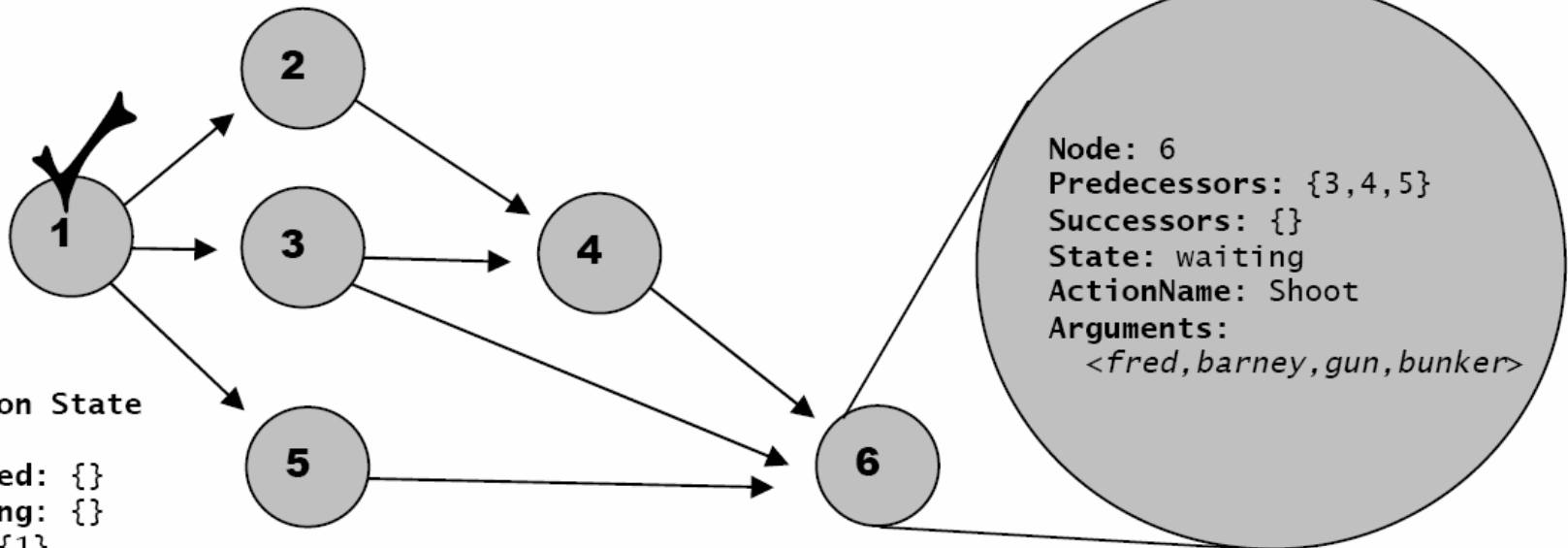
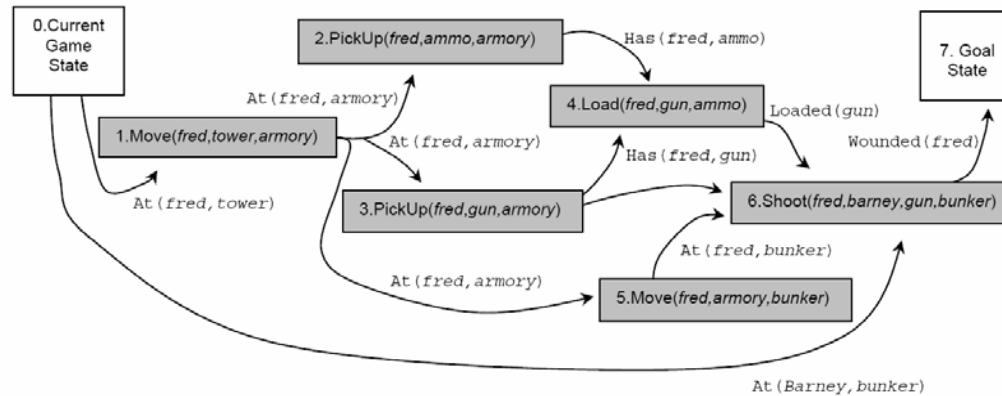
```
precondResult = CheckPreconds();
if (precondResult != -1) {
    reportPrecondFailure(precondResult);
    gotoState('Idle'); }
Body();
effectsResult = CheckEffects();
if (effectsResult != -1) {
    reportEffectFailure(effectsResult);
    gotoState('Idle'); }
reportActionSuccess();
```

Body() implements the operator's behavior, changing the game state according to the intended meaning of the operator.

The Executing State is identical across all action classes, and so is typically defined in the top-level Action class. It is included here for reference.

The Executing code first checks the action's preconditions. If all preconditions are met, then it runs the action's body. Next, it checks the action's effects. If all effects hold, then the action sends a message to the execution manager indicating that it has completed successfully. If an error is encountered along the way, the action sends an error report to the execution manager, facilitating re-planning.

Execution Management



Execution State

Completed: {}

Executing: {}

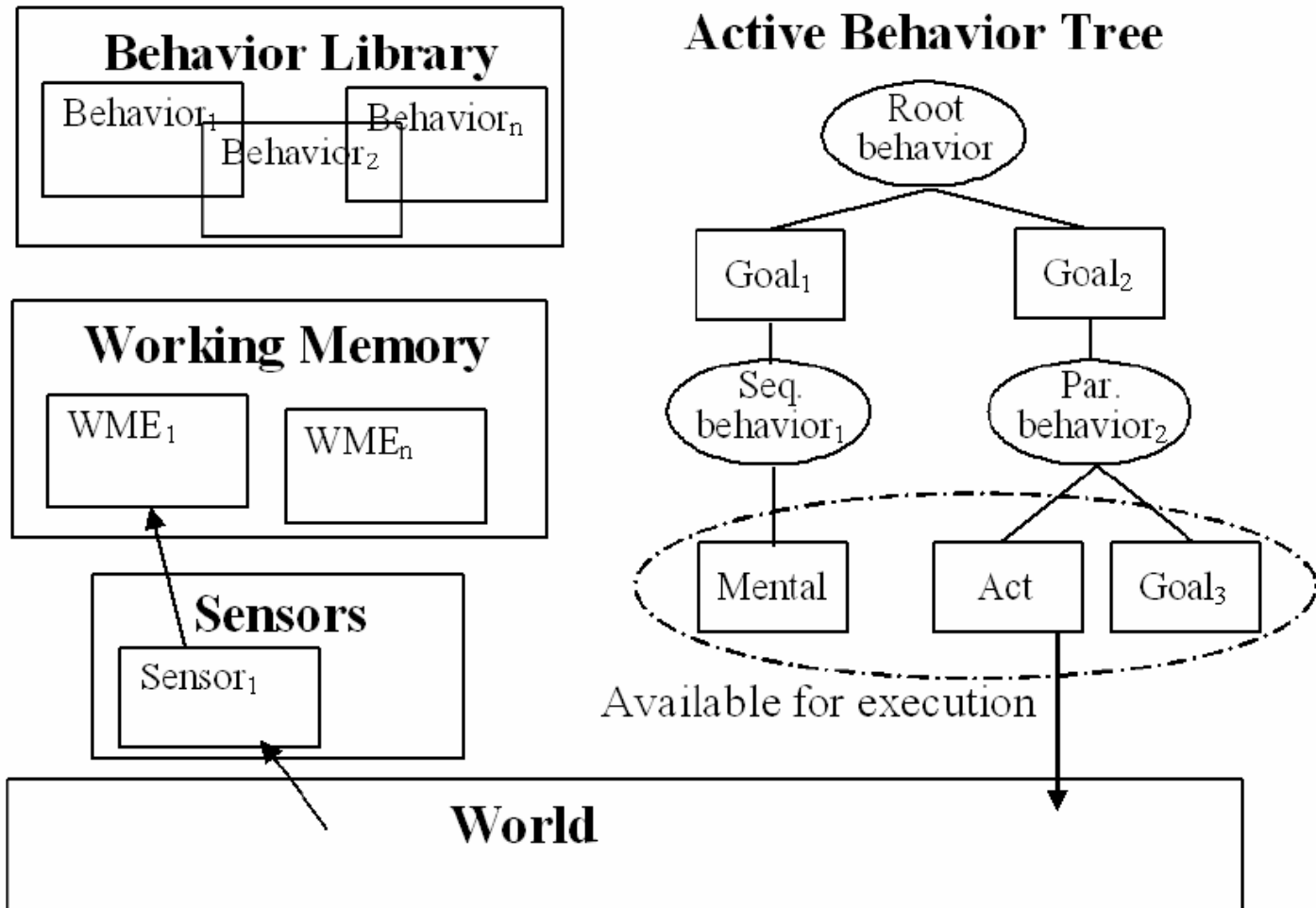
Ready: {1}

Waiting: {2,3,4,5,6}



no, I think it looks fine!

Façade Character Behaviors



Façade Action Example

```
sequential behavior AnswerTheDoor() {  
  WME w;  
  with (success_test { w = (KnockWME) }) wait;  
  act sigh();  
  subgoal OpenDoor();  
  subgoal GreetGuest();  
  mental_act { deleteWME(w); }  
}
```

```
parallel behavior YellAndWaitForGuestToEnter(int doorID) {  
  precondition { (CurrentTimeWME t :: startT)}  
  context_condition {(CurrentTimeWME t <= startT + 10000)}  
  number_needed_for_success 1;  
  with (success_test {{DoorOpenWME door == doorID}}) wait;  
  with (persistent) subgoal YellForGuest(doorID);  
}
```

```
sequential behavior OpenDoor() {  
  precondition {  
    (KnockWME doorID :: door)  
    (PosWME spriteID == door pos :: doorPos)  
    (PosWME spriteID == me pos :: myPos)  
    (Util.computeDistance(doorPos, myPos) > 100)  
  }  
  specificity 2;  
  // Too far to walk, yell for knocker to come in  
  subgoal YellAndWaitForGuestToEnter(doorID);  
}
```

```
sequential behavior OpenDoor() {  
  precondition { (KnockWME doorID :: door) }  
  specificity 1;  
  // Default behavior - walk to door and open  
}
```

Readings

- **[A Behavior Language: Joint Action and Behavioral Idioms](#)**
Michael Mateas and Andrew Stern Book chapter in [Life-like Characters. Tools, Affective Functions and Applications](#), eds. H. Prendinger and M. Ishizuka, Springer, 2004
- **[An architecture for integrating plan-based behavior generation with interactive game environments](#)**, by R. Michael Young, Mark O. Riedl, Mark Branly, A. Jhala, R.J. Martin, C.J. Saretto, *Journal of Game Development*, 1, 2004.