ROBOTICSLAB
Where technology happens

A Lightweight Finite State Machine C++ Library aimed at Seamless Integration with Robotic Middlewares

David Estévez, Juan G. Victores, Carlos Balaguer

destevez@ing.uc3m.es

Robotics Lab research Group
Universidad Carlos III de Madrid

Humanoids2016
IEEE-RAS International Conference on Humanoid Robots

Universidad
Carlos III de Madrid
www.uc3m.es

Outline

1. Introduction
2. Robot Devastation
3. StateMachineLib
4. Conclusions

Introduction

Robots are used nowadays in different areas:

- Industry, manufacturing
- Transportation
- Assistive technologies
- Military applications

Introduction

But robots can be also used for **entertainment...**

Introduction

Augmented Reality (AR) games can be enhanced with robots.

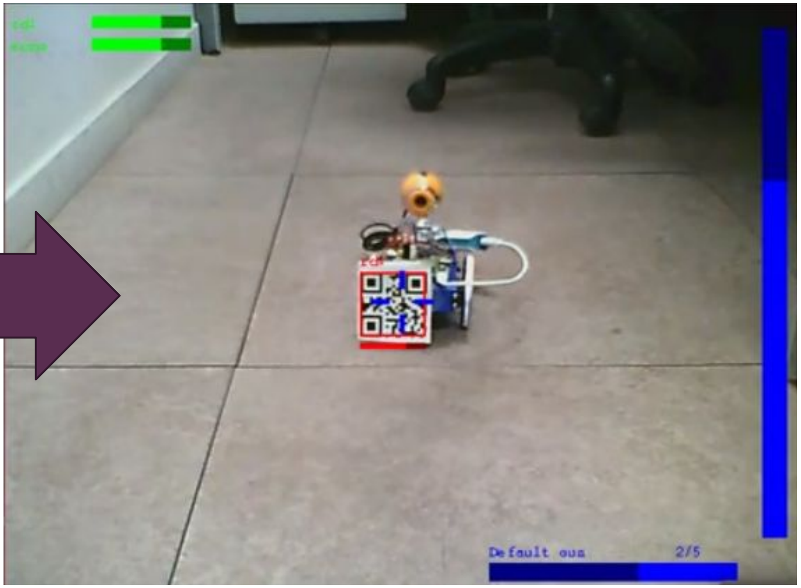
Limitations:

- Real world avatars -> real world issues (noise, latency, etc).

- Ambitious vs feasible

ROBOT DEVASTATION

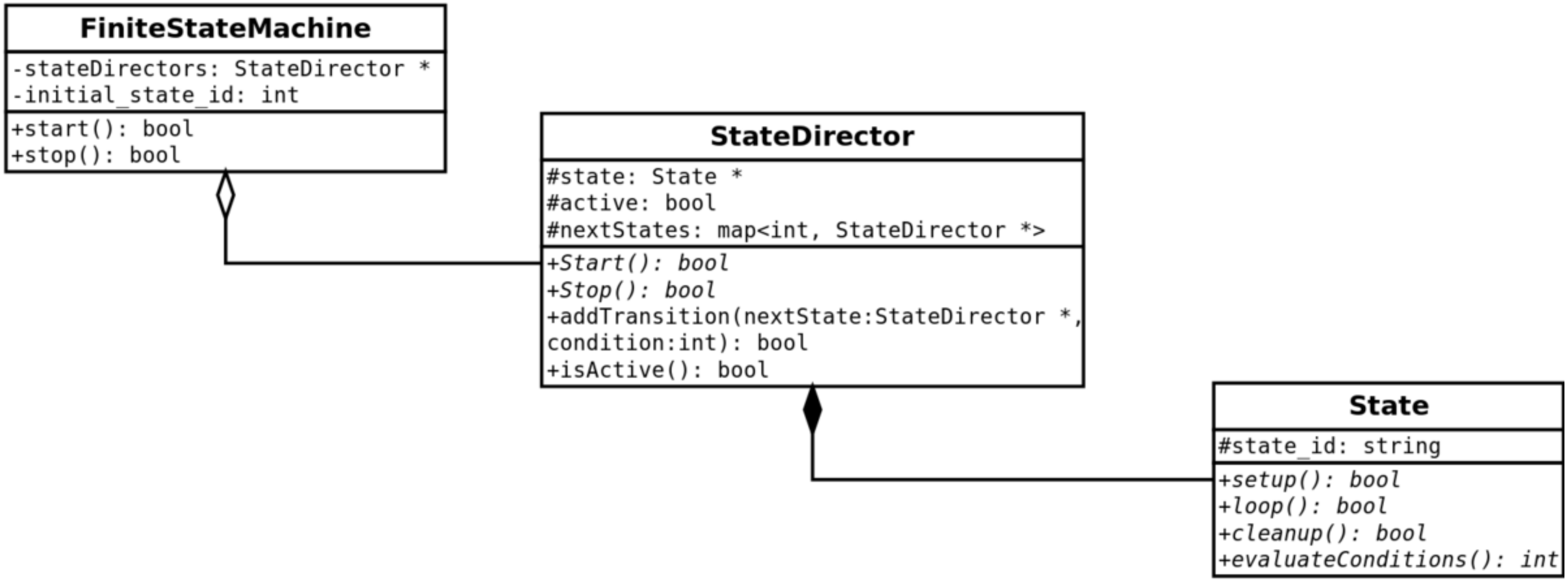
Real Robot Combat



Robot Devastation (Estevez, Victores, Morante & Balaguer, 2015)

StateMachineLib

rd::StateMachineLib main classes



FiniteStateMachine

```
-stateDirectors: StateDirector *
-initial_state_id: int
+start(): bool
+stop(): bool
```

StateDirector

```
#state: State *
#active: bool
#nextStates: map<int, StateDirector *>
+Start(): bool
+Stop(): bool
+addTransition(nextState:StateDirector *,
condition:int): bool
+isActive(): bool
```

State

```
#state_id: string
+setup(): bool
+loop(): bool
+cleanup(): bool
+evaluateConditions(): int
```

StateMachineLib

State: base class for each FSM state.

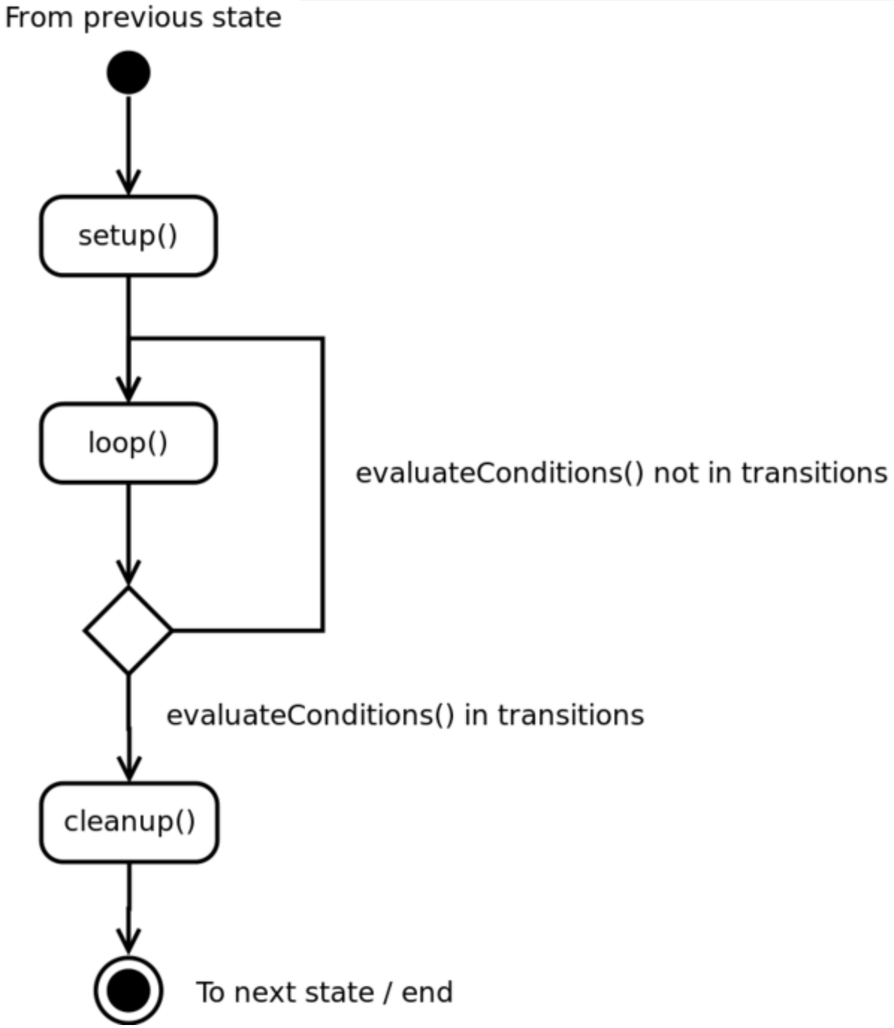
- setup()
- loop()
- cleanup()
- evaluateConditions()

StateMachineLib

StateDirector:

Controls FSM flow

- Different possible implementations: YARP, C++11, pthread, etc


From previous state
setup()
loop()
evaluateConditions() not in transitions
evaluateConditions() in transitions
cleanup()
To next state / end

StateMachineLib

FiniteStateMachine: wraps the states with a nice interface to manipulate the FSM

- **StateMachineBuilder** class is provided to encapsulate the creation of a FSM from the State objects.

Conclusions

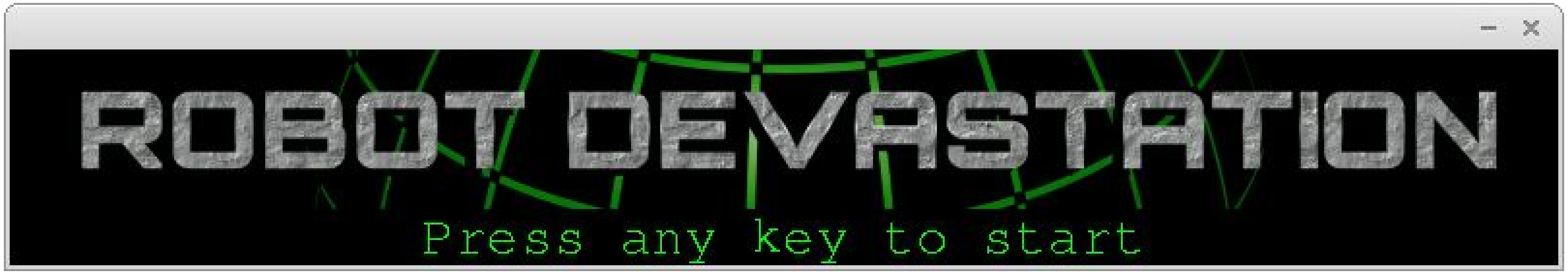
Robot Devastation game FSM Implementation

4 states:

- **Init**: game initialization and server logging
- **Game**: game main screen and gameplay
- **Dead**: dead screen and wait for respawn
- **Exit**: server logout and game cleanup

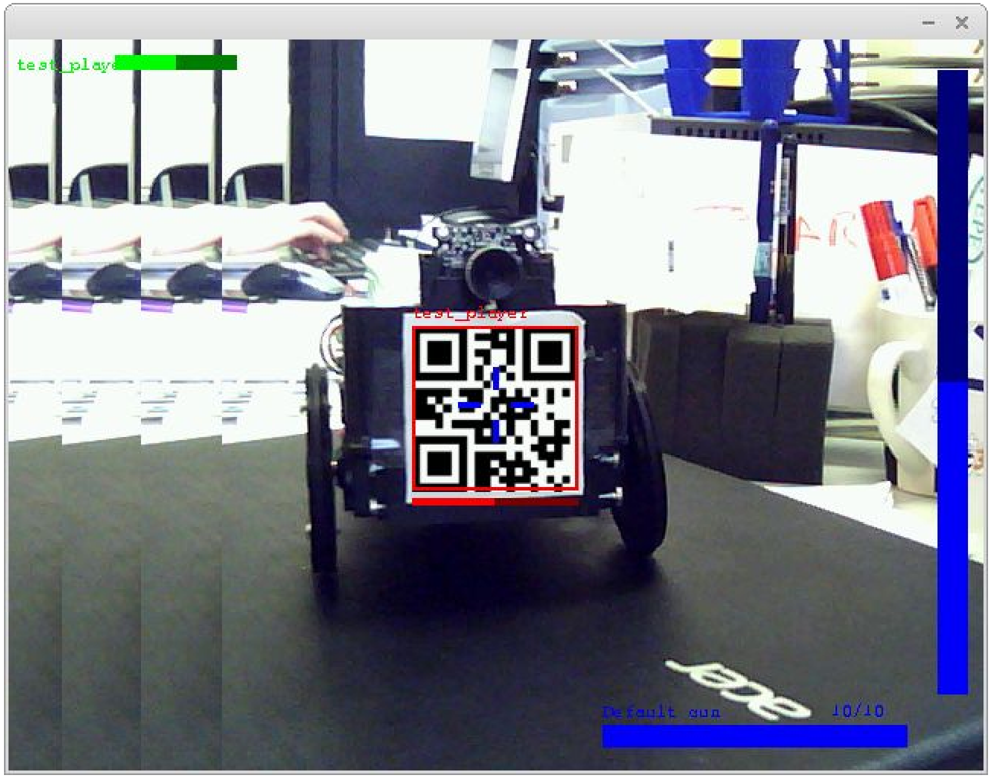
Conclusions

Init: game initialization and server logging



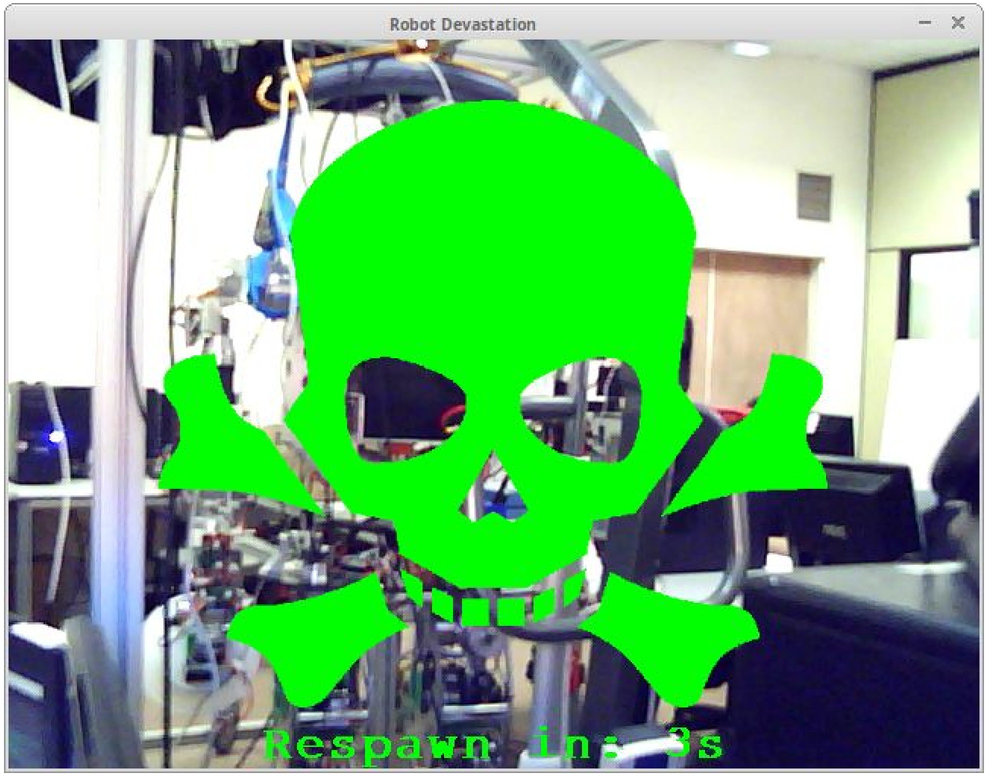
Conclusions

Game: game main screen and gameplay



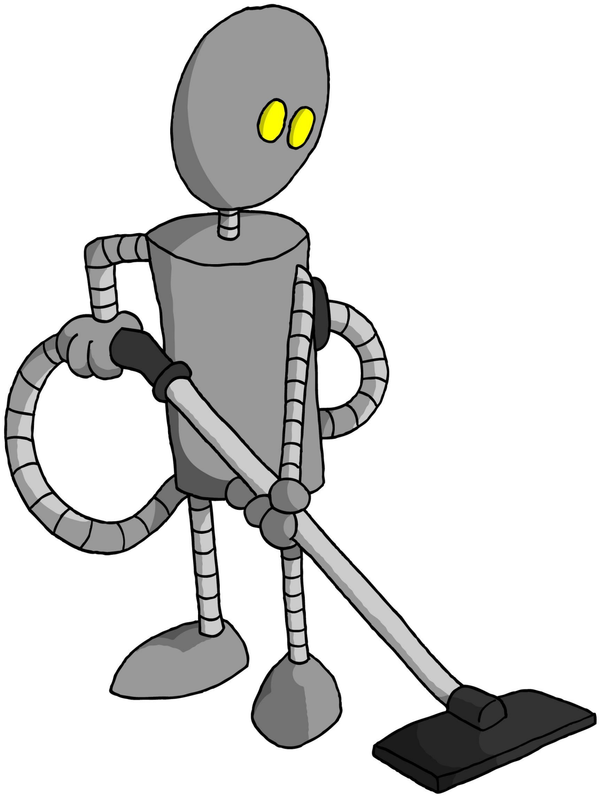
Conclusions

Dead: dead screen and wait for respawn



Conclusions

Exit: server logout and game cleanup



src: https://wemovechicago.com/moving-cleaning-checklist/

Join us!

- Source code available at: https://github.com/asrob-uc3m/robotDevastation
- Check out our Wiki too: http://asrob.uc3m.es/index.php/Robot_Devastation

ROBOTICSLAB
Where technology happens

A Lightweight Finite State Machine C++ Library aimed at Seamless Integration with Robotic Middlewares

Thank you!

David Estevez, Juan G. Victores, Carlos Balaguer

Humanoids2016
IEEE-RAS International Conference on Humanoid Robots

Universidad
Carlos III de Madrid
www.uc3m.es