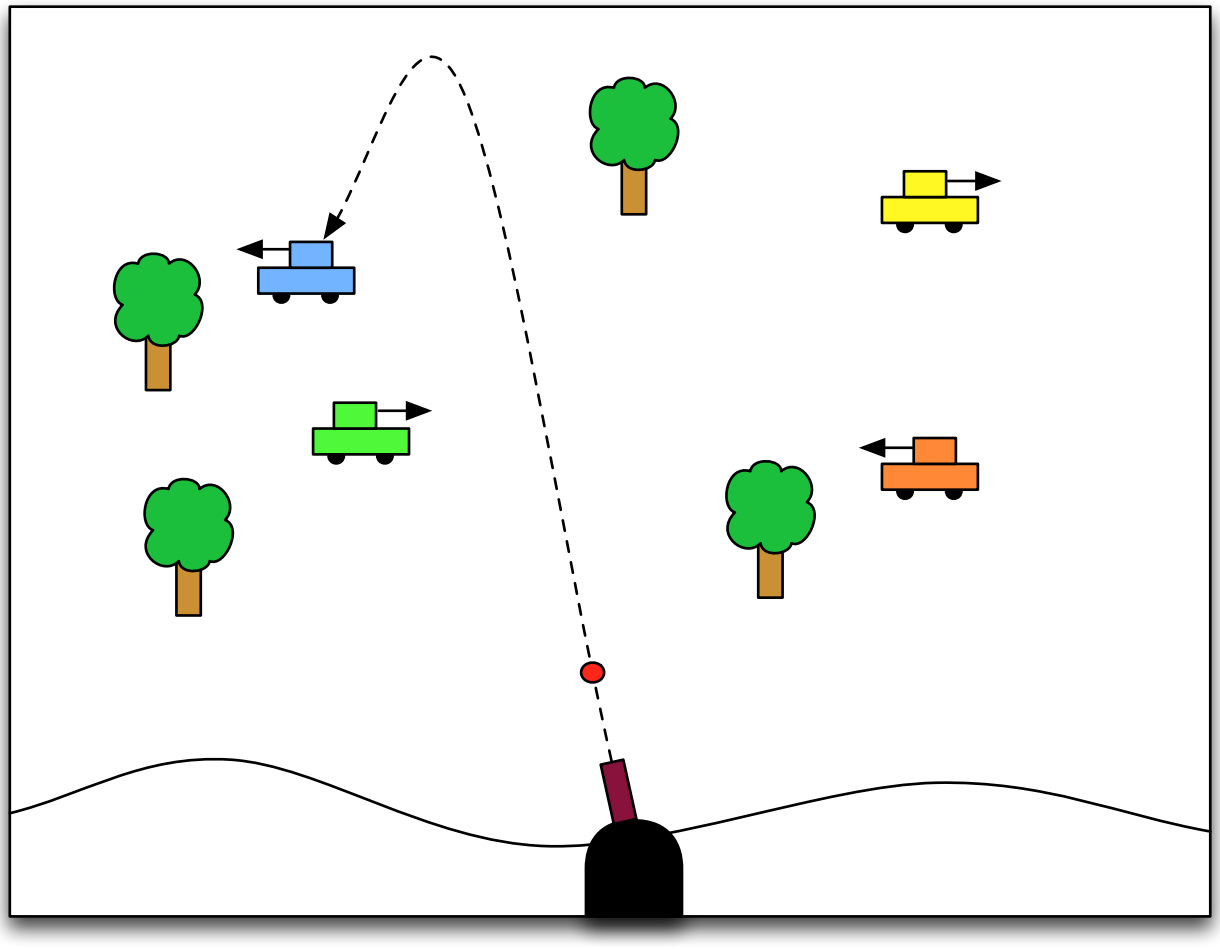
Realtime Interactive Systems
Final Task

Würzburg University - HCI Group

Winter Term 2013/14



Task Description

To complete this course you have to implement and present a Realtime Interactive System, utilizing the knowledge you gained during the lecture.

In the optimal case, your application contains three or more simulation engines: A renderer, a physics engine and an artificial intelligence (AI) component. By connecting these components a game depending on their functionality shall be created.

The game's content is depicted in the figure on the title page: The player is controlling a small cannon, which can shoot physically simulated cannonballs. In front of the cannon vehicles are moving around, which are controlled by the AI component. There should be some obstacles in the area where the vehicles move.

The player's task is to destroy all vehicles, whereas the AI component has to avoid collisions with cannon balls (which should be removed after two or three rebounds). Furthermore, the AI controlled vehicles shall pick up objects that are randomly placed on the map. The player looses if the AI manages to collect all object on the map.

You may extend or alter this game, as long as its main features are maintained. If in doubt, don't hesitate to ask if your modifications would be ok.

You are encouraged to use the akka actor library (available at `http://akka.io`). Furthermore, you may use any software library that does not implement one of the features which are part of the grading (see next section). For example, using the VRPN library to connect sensors to your application is fine, but using an external physics engine is not. The only exception to this is the opportunity to use an external renderer (e.g. the jMonkey engine, except for its physics implementation, of course, or your software from the interactive computer graphics course).

Remember: This exercise is **not** about fancy graphics and ultra-intelligent AI but the interplay of the implemented components as well as the RIS's internal structure.

Grading

Your program will be graded according to table 2. The listed values denote the maximum number of points which can be achieved with the respective task. Partial completion of the task or unclean implementation will result in less points than the value from the table. Table 1 shows the mapping from the achieved points to the final grade.

Note: It is essential to provide a good presentation of your software, since only you could point out the specifics of your implementation, especially important features which can not be seen since they compose the core system.

Points	100	95	90	85	80	75	70	65	60	55	50
Grade		1.0	1.3	1.7	2.0	2.3	2.7	3.0	3.3	3.7	4.0

Table 1: Points to grade mapping

Points	Feature Description	Notes
20	RIS Core System	
20	Event System	
10	Presentation	talk and demonstration
10	Renderer	may use an external component here
10	Physics Engine	collisions & forces (for spheres and planes)
10	AI Component	Path planning to evade cannon balls
5	Animations	e.g. pick up animation
5	Level Design	**not** graphically rich but suited to show interaction of components
5	User Interface	
5	Box to Box Collisions	
up to 15	Own Ideas	ask if you have questions

Table 2: Achievable points

Milestones (suggestion)

- [ ] create time schedule ende April
- [ ] implement event system
- [ ] implement data synchronization mechanism
  - [ ] create representation for spheres
  - [ ] create representation for planes
- [ ] implement basic(!) renderer features
  - [ ] create representation for spheres and planes
- [ ] Implement physics engine
  - [ ] create internal representation for planes and spheres
  - [ ] implement collision detection
  - [ ] implement collision resolution
  - [ ] implement lifetime control
- [ ] create application with cannon
  - [ ] design cannon (no physical representation)

- [ ] react on user input
- [ ] create level (ground plane, maybe one obstacle)

[ ] implement AI component

- [ ] implement data structure for level representation (e.g., waypoint graph)
- [ ] implement pathfinding algorithm
- [ ] implement method to modify graph to avoid collisions (with obstacles, other vehicles and cannon balls)

[ ] implement vehicles

- [ ] create physical, graphical and AI component's representation
- [ ] make AI component control vehicles
- [ ] make vehicles decelerate instead of abruptly changing directions

[ ] implement application logic of final game

- [ ] add collectables
- [ ] remove vehicle on collision with cannon ball
- [ ] make AI component listen for new balls (and adjust paths)
- [ ] implement game end check

[ ] final bug fixing and beautifications