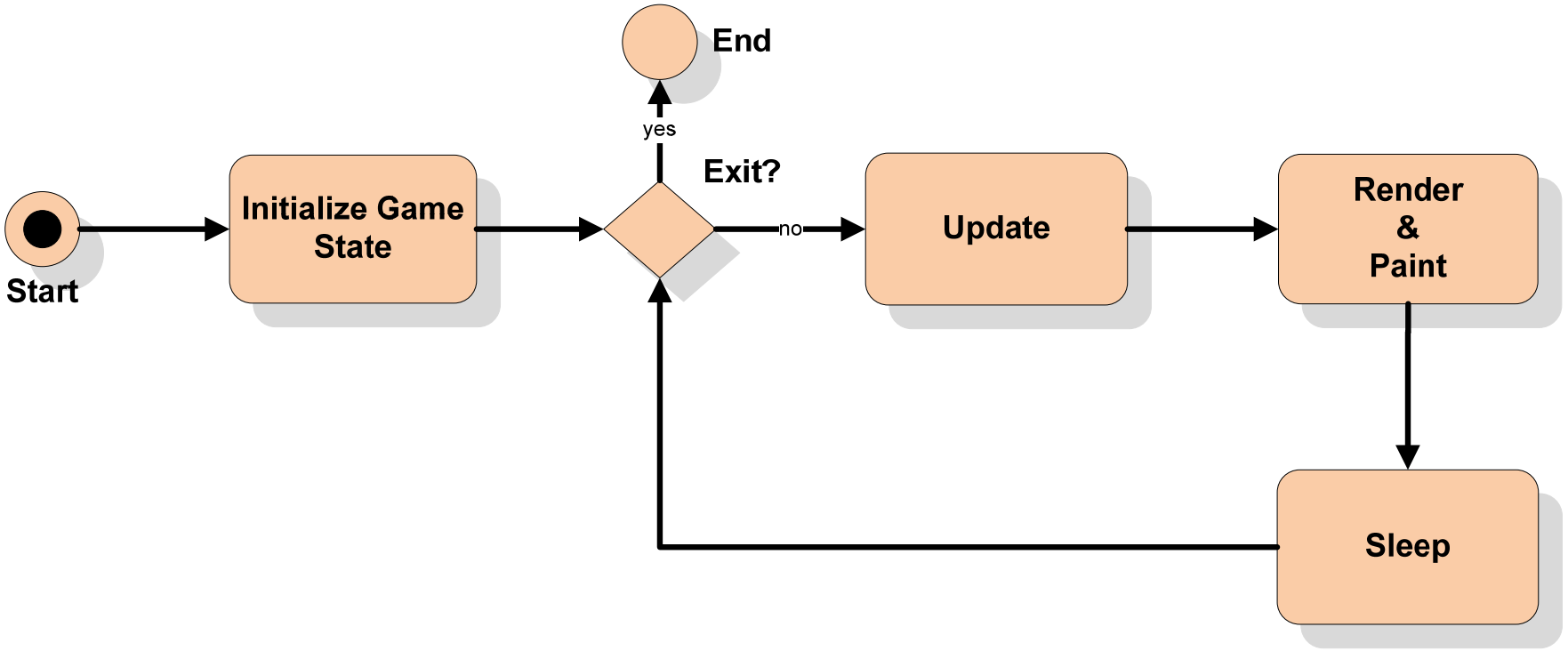
VK Computer Games

Mathias Lux & Horst Pichler
Universität Klagenfurt

Agenda

- Collision Detection
- Game Framework
  - Sprites
  - Sprite-behavior
  - Bricks
- Short Introduction into AI
- Your Game … some advices

The Game Loop Revisited

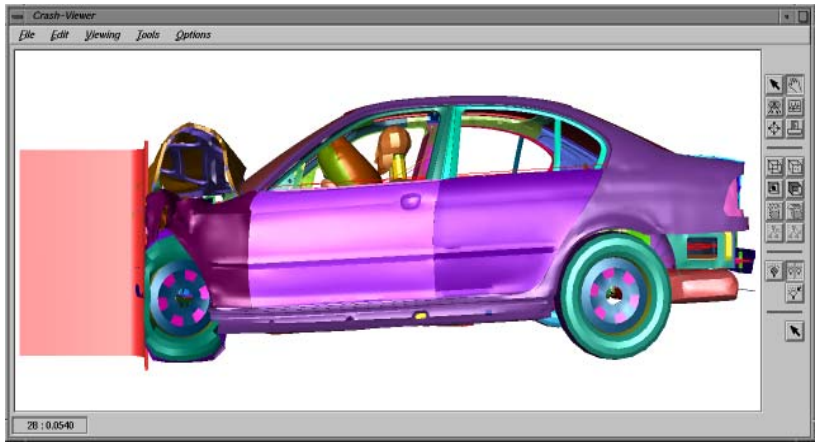


➔ Double Buffering with repaint-method

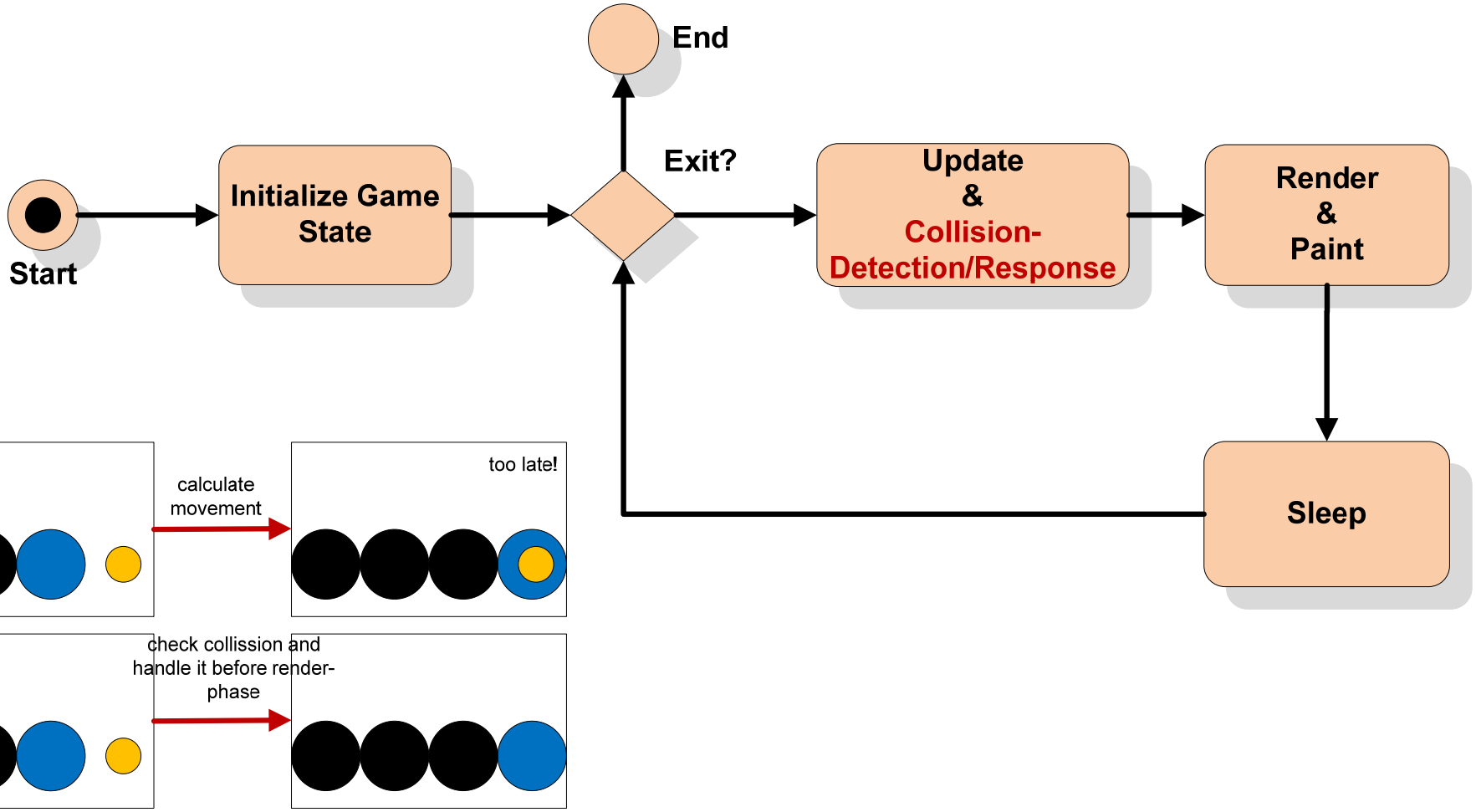
Collision Detection

- Detect if two objects try to occupy the same space at the same time (overlap)

- Needed in
  - physical simulations
  - computational geometry
    - CAD
    - GIS
  - robotics
  - computer games



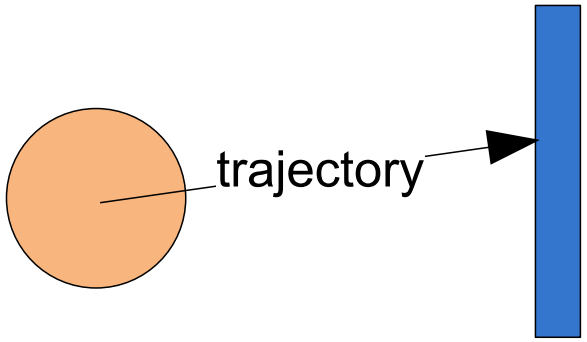
Game Loop + Collision



Remark: example assumes, that there is no z-axis

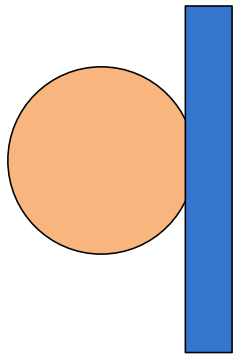
Detection Types

- a priori
  - predict upcoming collision
  - objects never really penetrate
  - respond to collision before it occurred

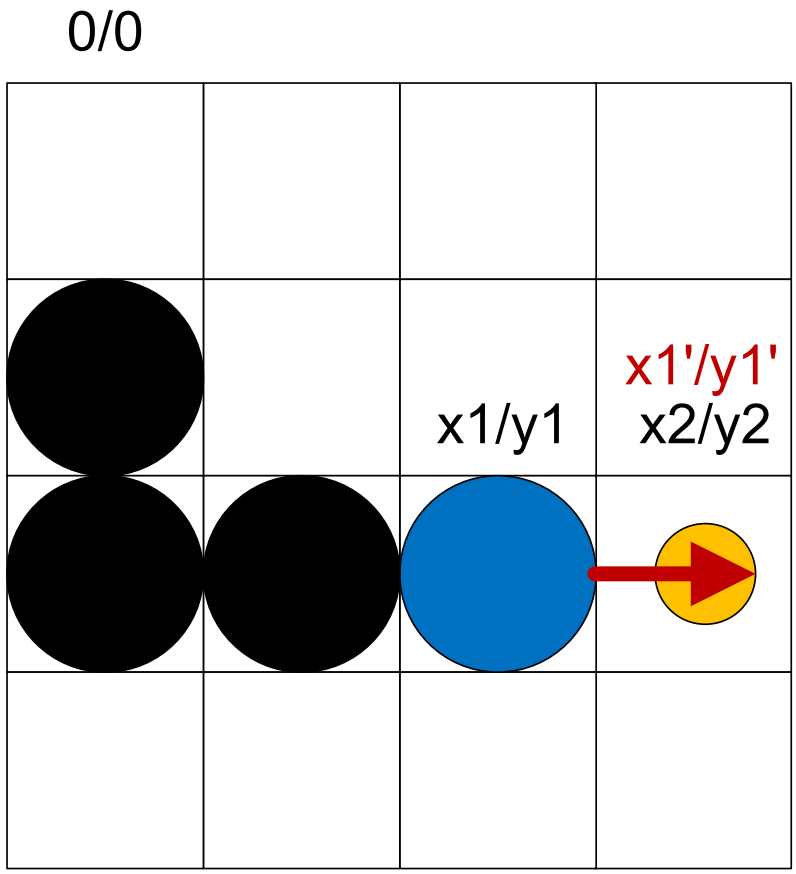


- a posteriori
  - advance movement by small time step
  - check if collision occurs
  - respond to collision after it occurred

  ➔ much easier; used in games



2D Games with Static Arrays

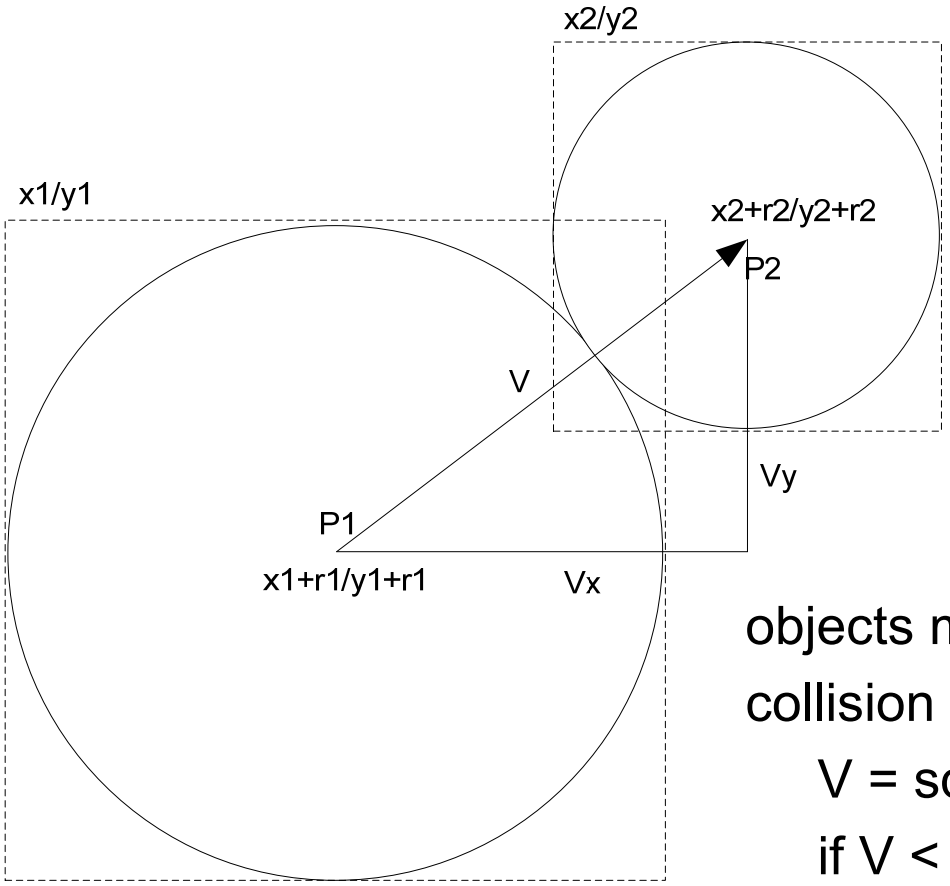


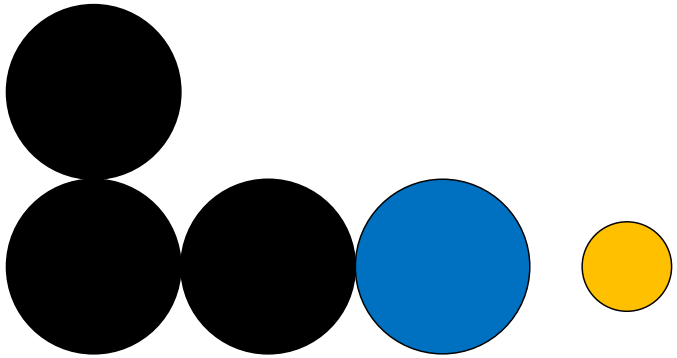
Object moves 1 field per round

Collision:

(x1‘ == x1) and (y1‘ == y2)

Circles





objects move n pixels per loop

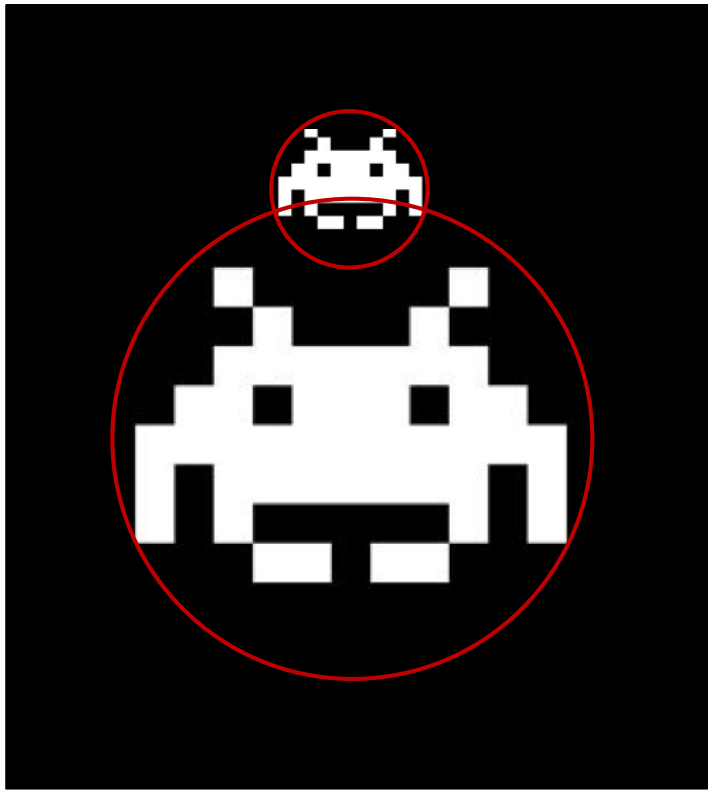
collision detection with Pythagoras:

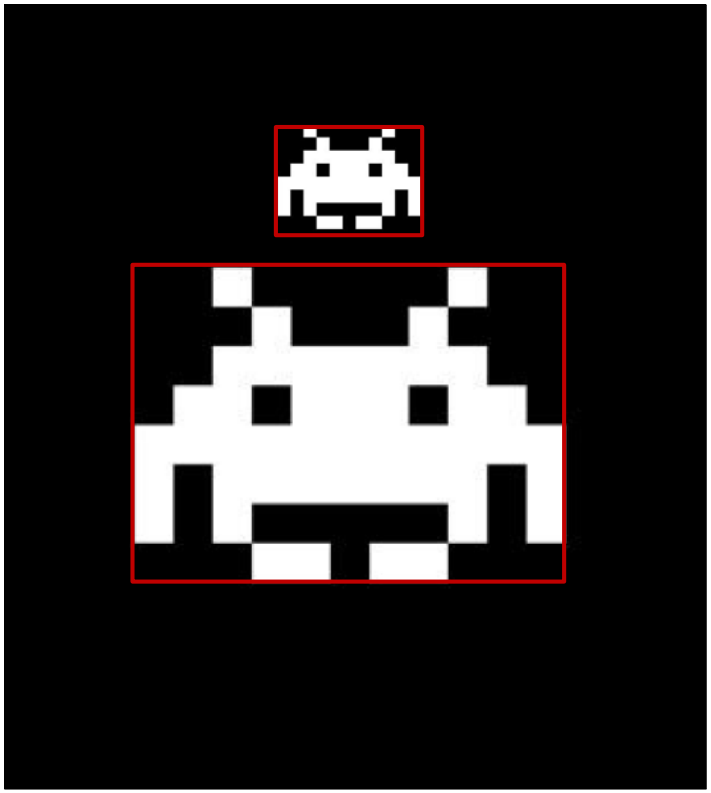
$V = \sqrt{(x2-x1)^2 + (y2-y1)^2}$

if V < r1 + r2 ➔ collision

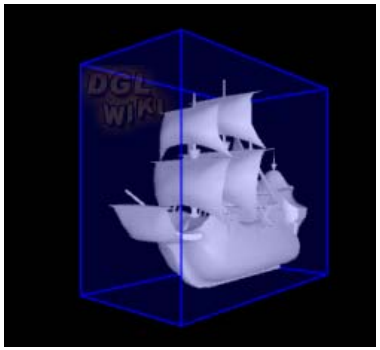
Problems with Circles

- Easy to calculate, but …
- Bounding box

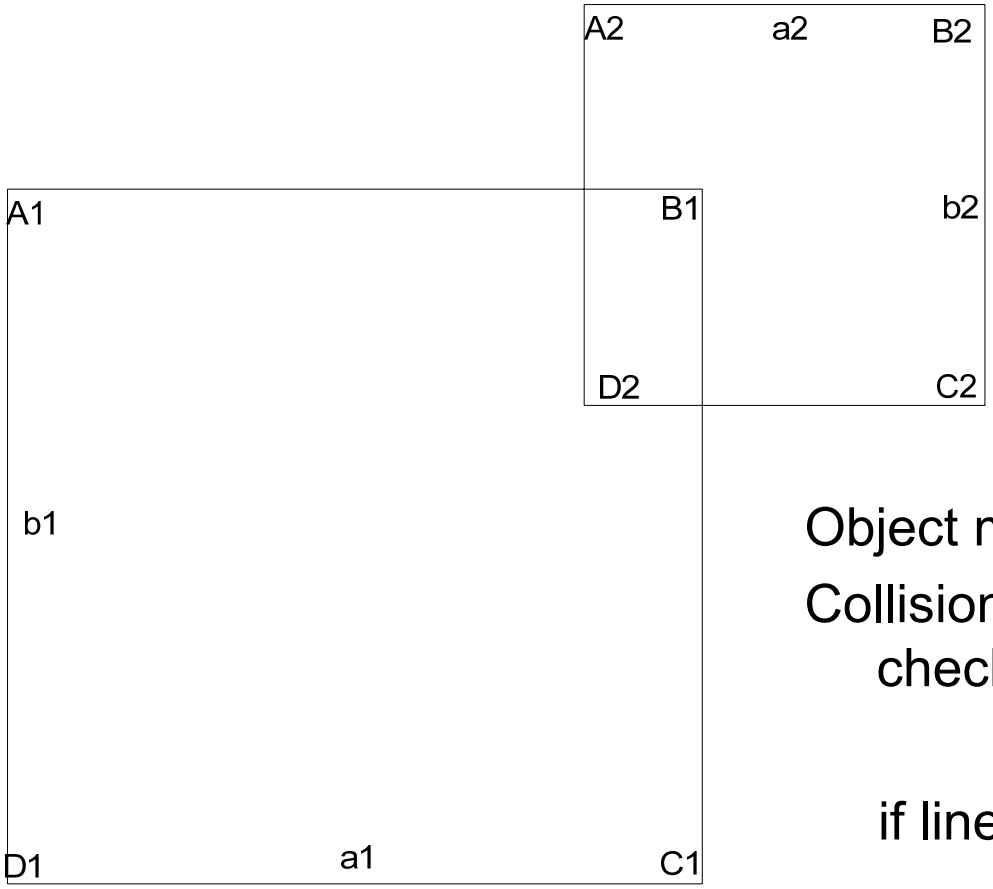




Bounding Volume



Rectangles



Object moves n pixels per loop

Collision detection: „inelegant“ approach
check possible combinations of lines
[A1B1xA2D2, A1B1xB2C2, …]
if lines cut ➔ collission

Example

Collision detected
B1C1xA2D2

- bounce left (if jumps)
- stop x-movement

Collision detected
C1D1xA2B2

- new ground-level
- stop y-movement

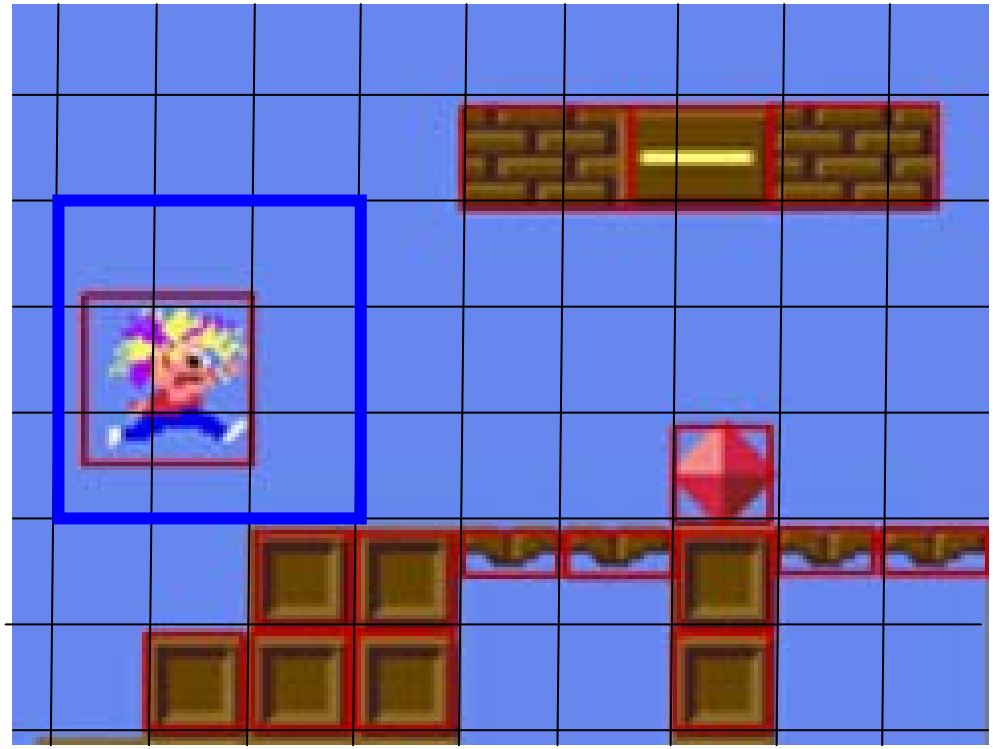
Collision detected
C1D1xA2B2

- bounce down
- reverse y-movement

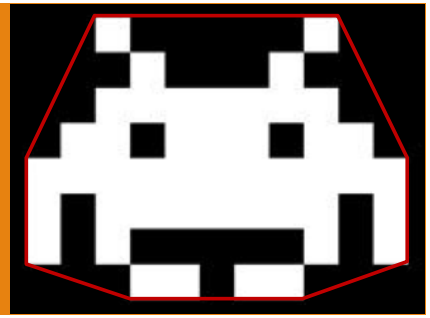
[The Great Giana Sisters, Rainbow Arts: http://www.youtube.com/watch?v=zrRHAisdIRg]

Example

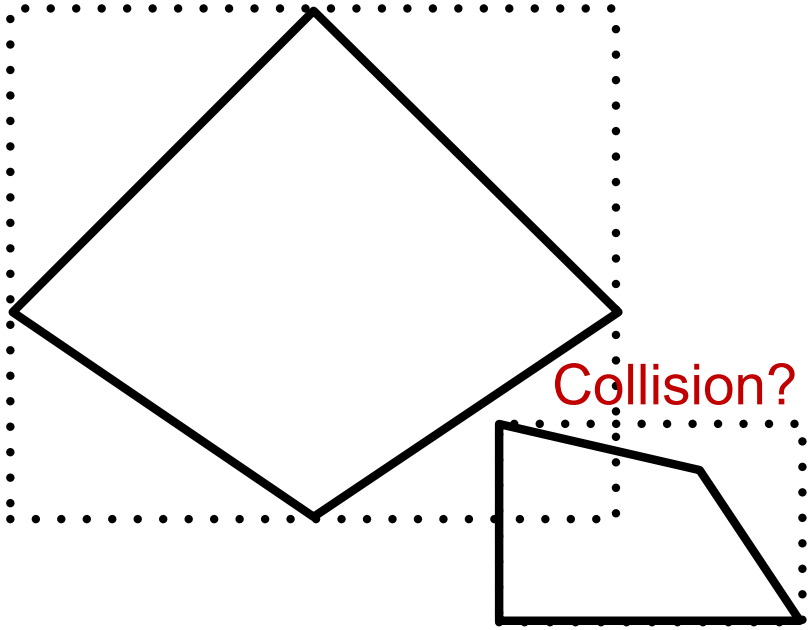
- Optimization issues
  - check only objects in rows and/or columns of relevant x/y-coords
  - check moving objects only (?)

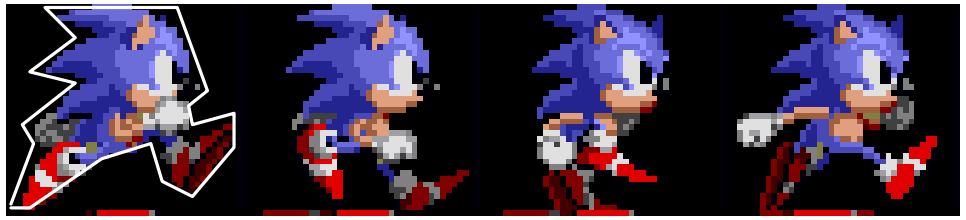


Convex Polygons



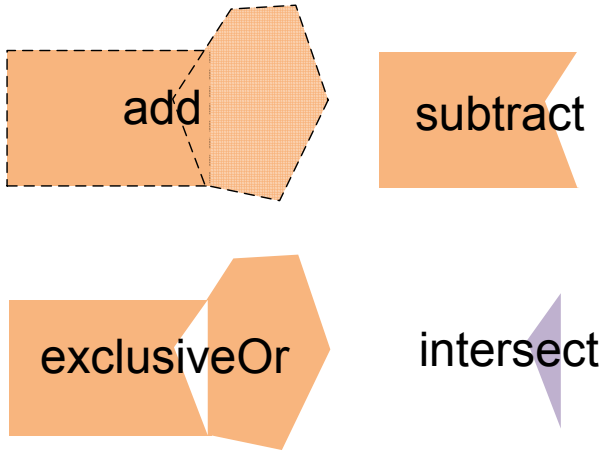
- Collision detection:
  - bounding box
    - imprecise
  - checking every possible pair?
    - what exactly means “possible”?
    - complex polygons meshes!
  - prerequisites:





Useful Java Methods

- Interface Shape (objects with geometric shape)
  - boolean: contains(Point2D p)
  - boolean: contains(Rectangle2D r)
  - Rectangle2D: getBounds2d()
  - PathIterator: getPathIterator(AffineTransformation t)
- Implementing classes
  - Area
  - Polygon
  - Rectangle
  - Line2D
  - CubicCurve2D, QuadCurve2D
  - …
- Polygon
  - addPoint(int x, int y)
- Area
  - constructor: Area(Shape s)
  - Rectangle2D: getBounds(Area a)
  - boolean: isPolgonyal()
  - void: add(Area a)
  - void: subtract(Area a)
  - **void: intersect(Area a)**
  - void: exclusiveOr(Area a)



➔ collision detection with intersect

➔ but: no projection vector!

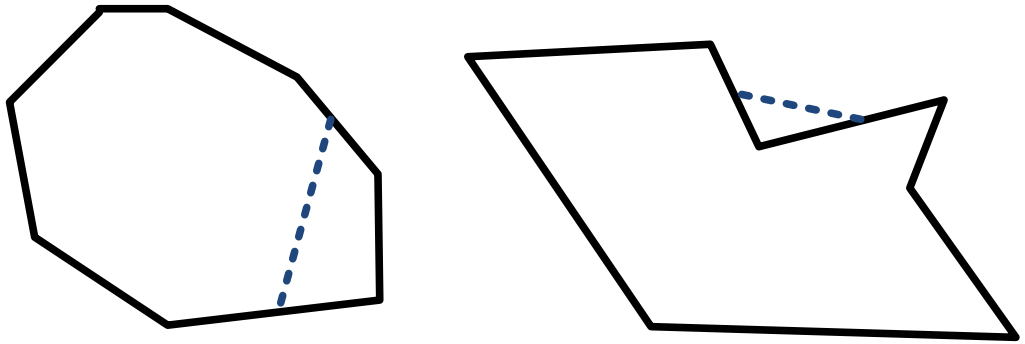
Example


```
public static boolean detect(Shape s1, Shape s2) {
    long t1=System.nanoTime();
    Area a1 = new Area(s1);
    a1.intersect(new Area(s2));
    boolean colide = false;
    if (!a1.isEmpty()) {
        colide = true;
        micros = (System.nanoTime()-t1)/1000;
    }
    return colide;
}
```

call: detect(polygon1, polygon2)

Separating Axis Theorem

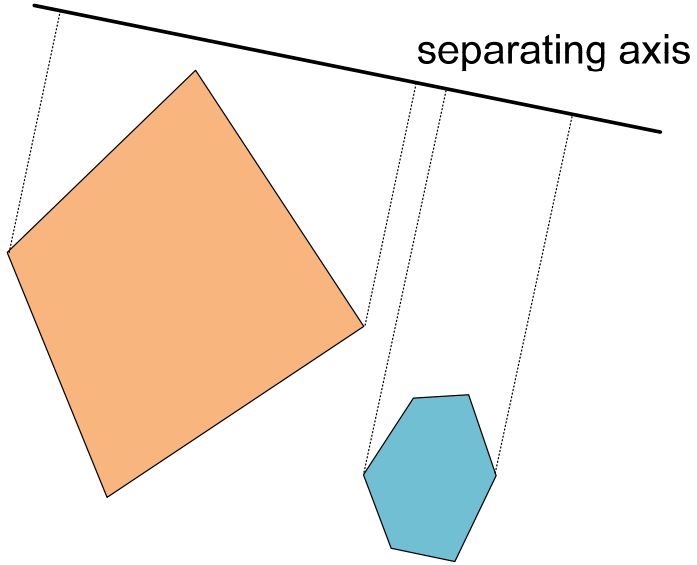
- Intersect-method provides no projection vector
  - ➔ separating axis theorem
- Convex shape: an object is **convex** if for every pair of points within one object, every point on the connecting straight line is within the object



convex non-convex

Separating Axis Theorem

- Given two convex shapes, if we can find an axis along which the projection of the two shapes does not overlap, the shapes don't overlap. [Herman Minkowski, 1864-1909]



Collision detection:

no separating axis ➔collision

or:

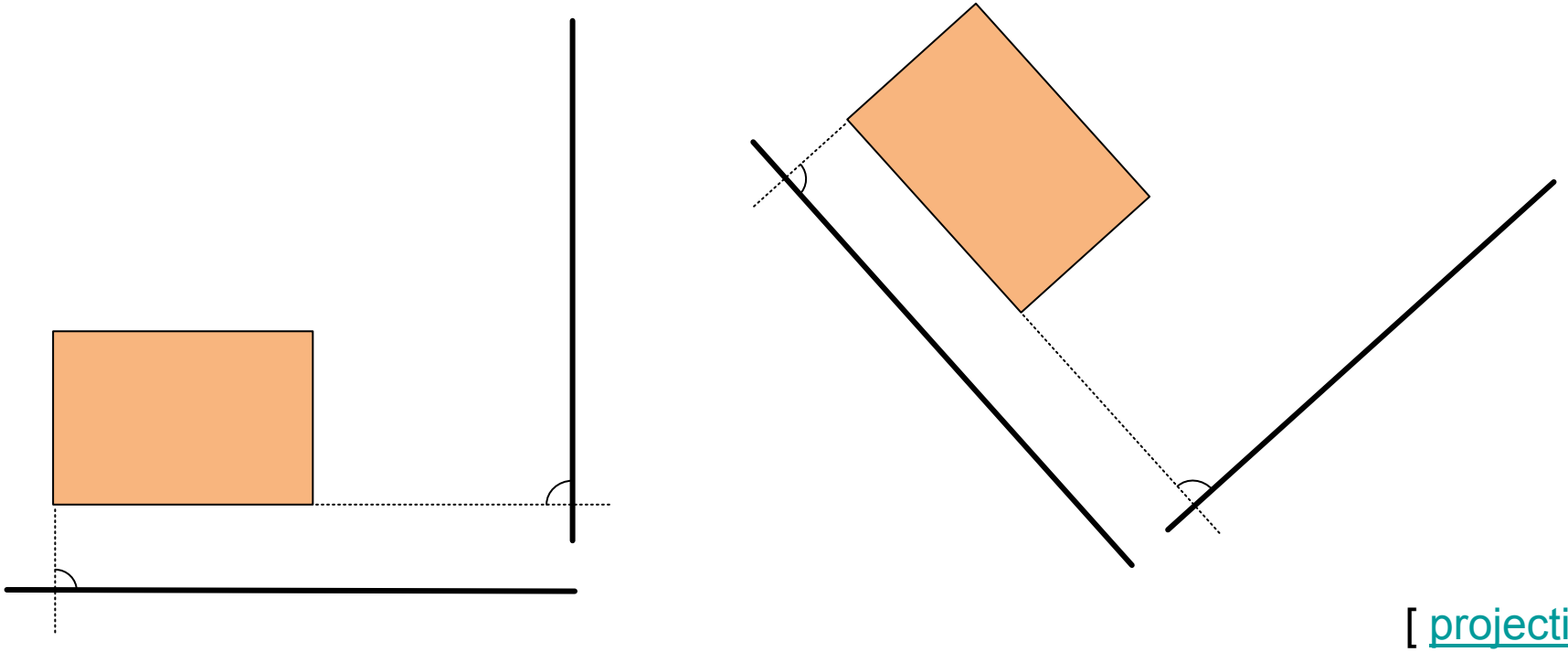
1 separating axis ➔ no collision

problem:

infinite number of axis to test?

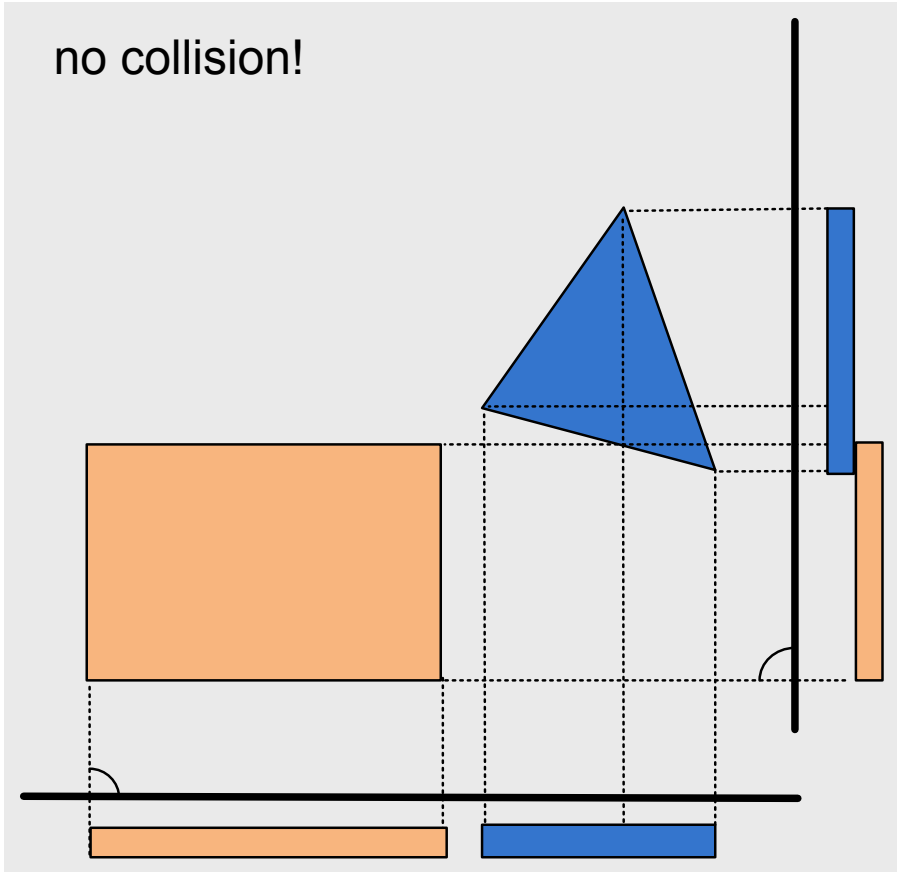
Separating Axis Theorem

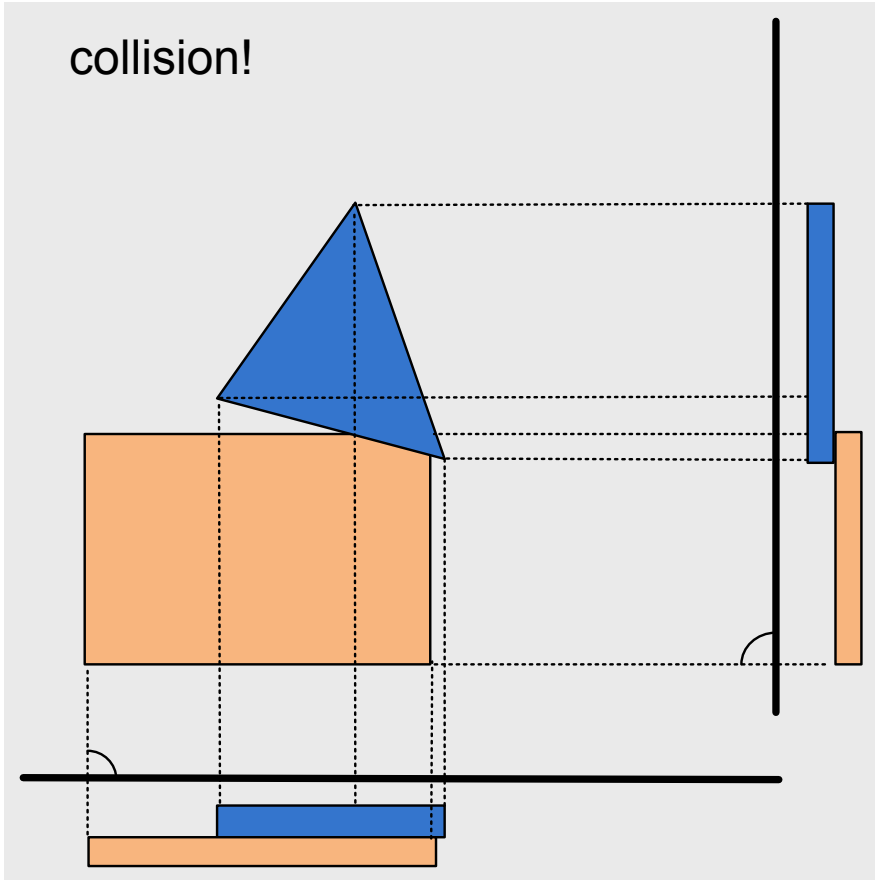
- For convex polygon meshes: each faces normal is used as an axis to test for separating axis



[projection]

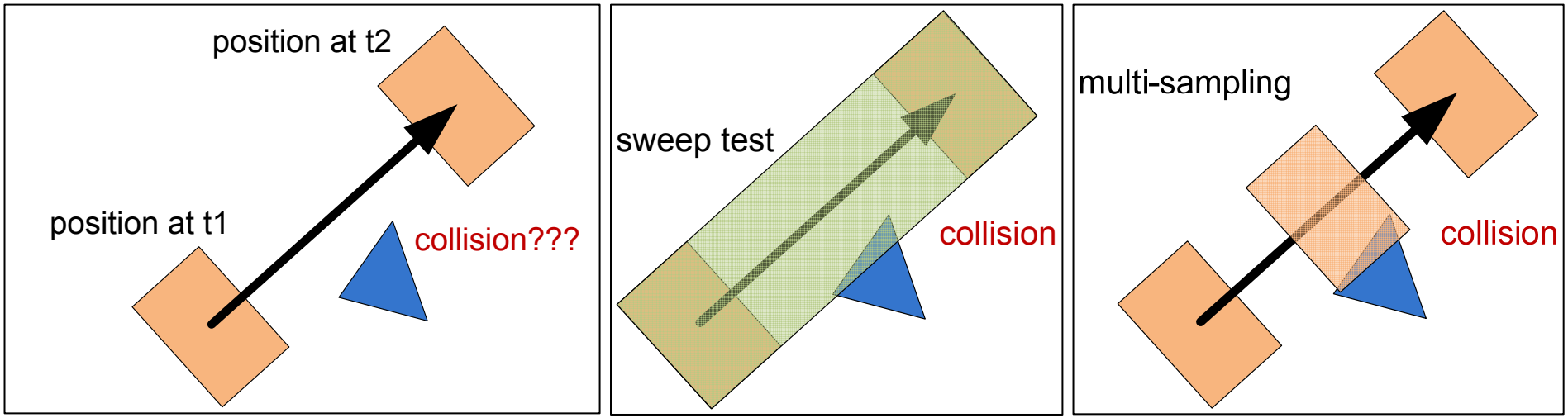
Collision Detection with SAT





[collision]

Fast Moving Objects

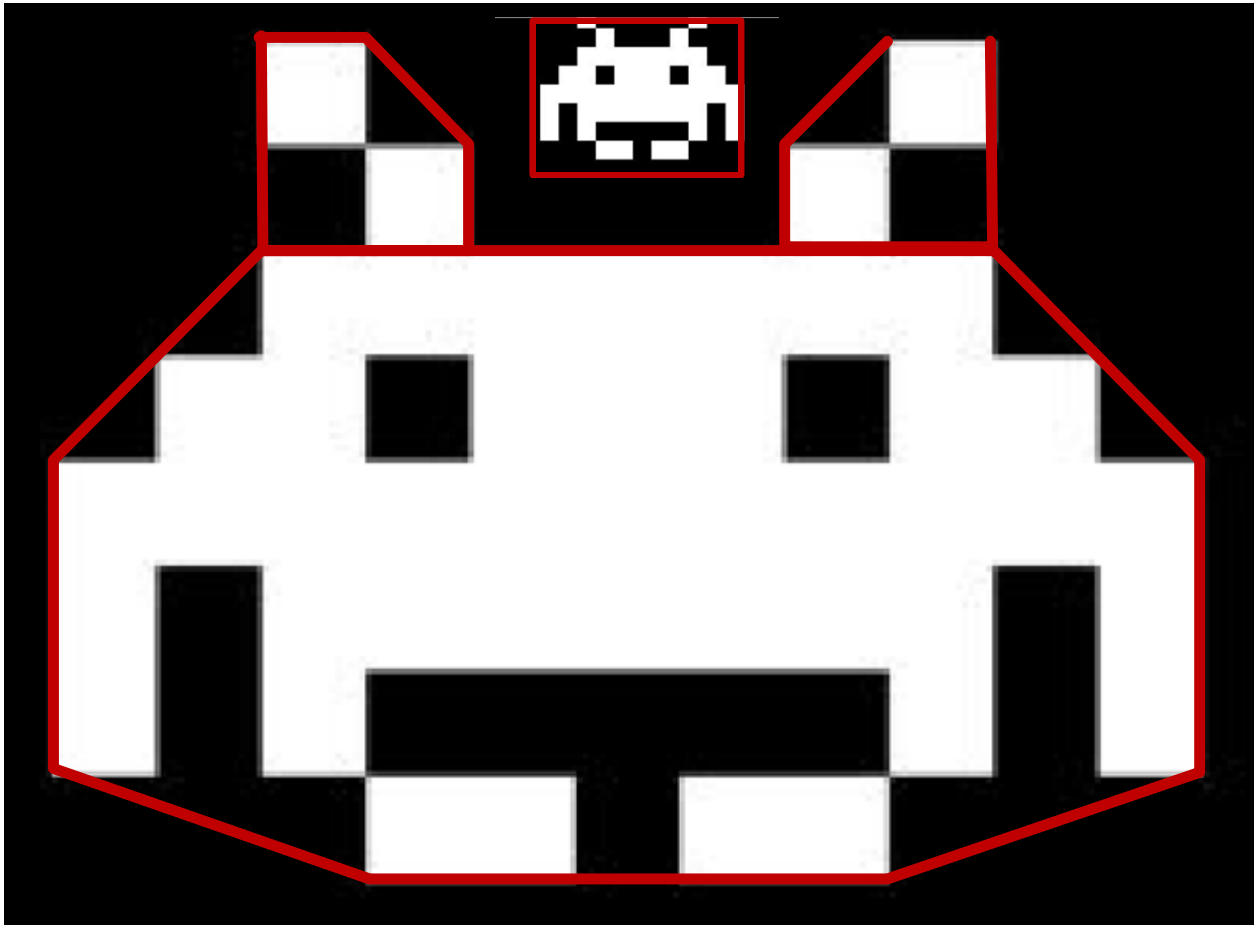


- Sweep test
  - create a new polygon along the trajectory
  - test collision with the new polygon
- Multi-sampling
  - create additional polygons in between
  - test for each of them

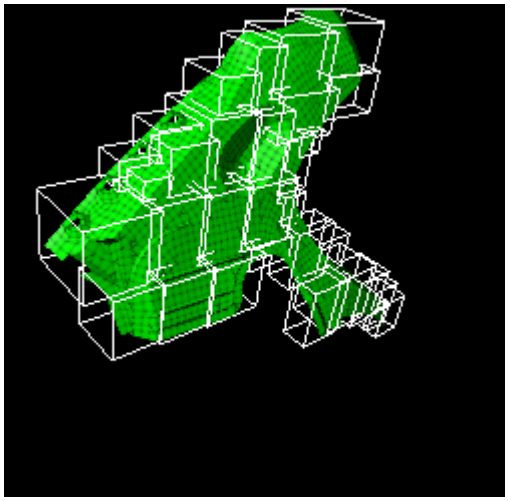
Is this sufficient?

R-Type, Irem[http://www.youtube.com/watch?v=xPv5lqpA4c4&feature=related]

Multiple Bounding „Boxes“



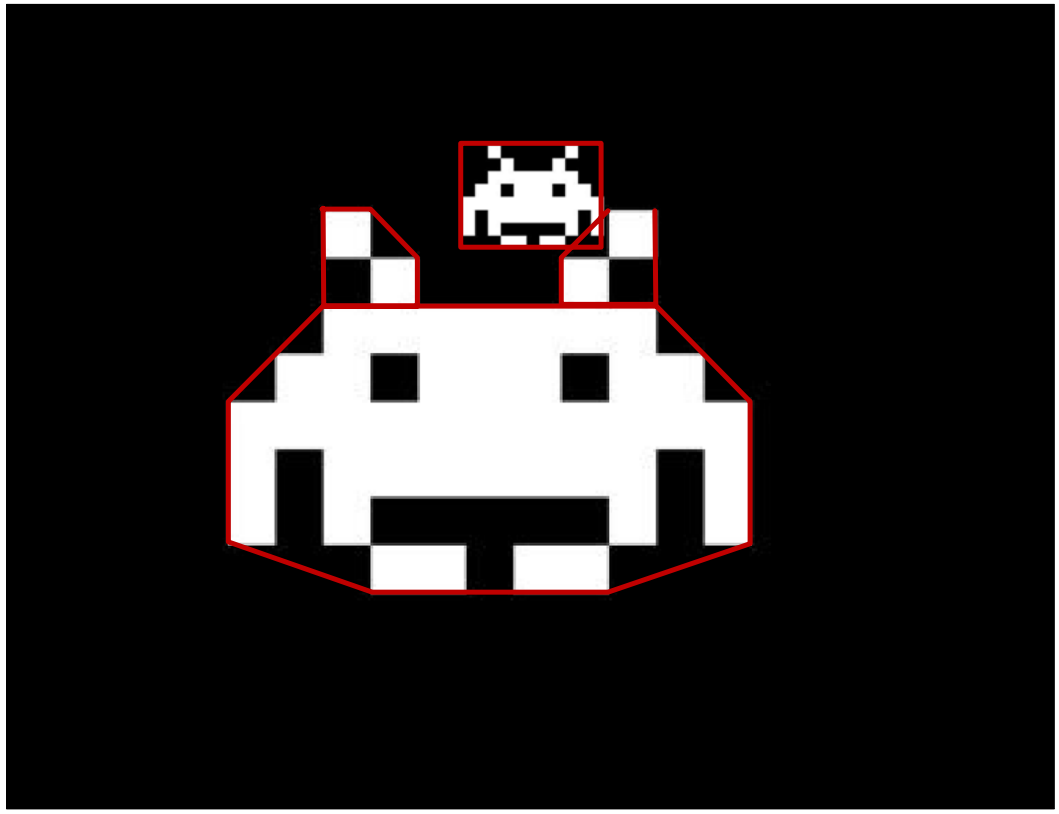
Big sprite is non-convex, therefor compose it from several convex polygons



… Bounding Volumes

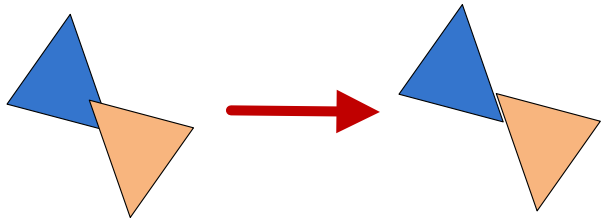
Collisions on the Pixel-level

- to avoid collision „over-detection“ we need an even more accurate method
  - first test on polygon-level
  - then test on pixel-level



Collission Response

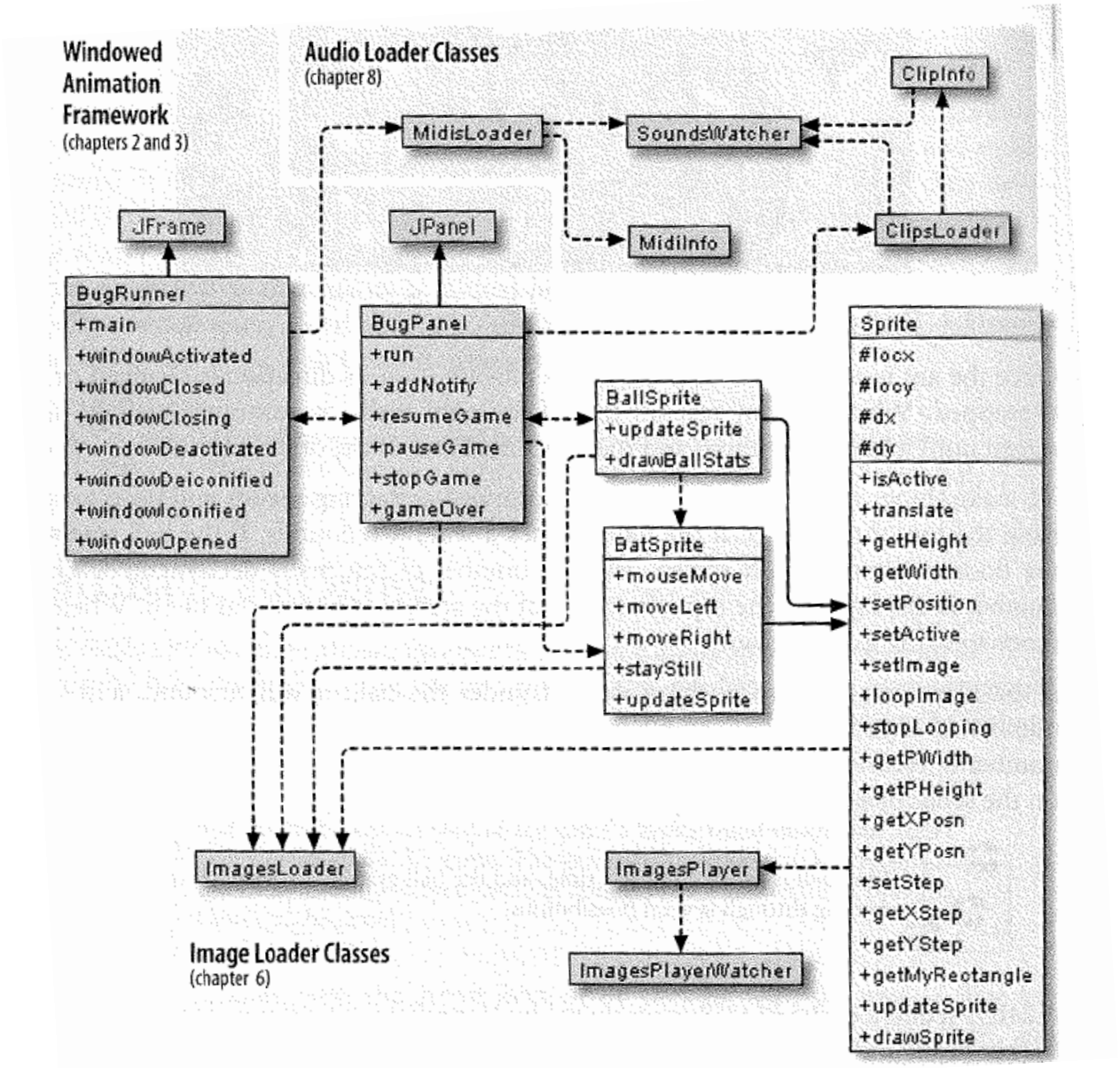
- Change the object‘s status (e.g. to „exploding)
- Remove one or both objects
- Projection methods
  - find the smallest possible displacement
  - direct modification of object positions



- Penalty force methods [collision.html]
  - use spring forces to pull object out of collision
  - modify the objects acceleration / direction
- Impulse based methods
  - use changes in velocity to prevent interpenetration (e.g. stop object)

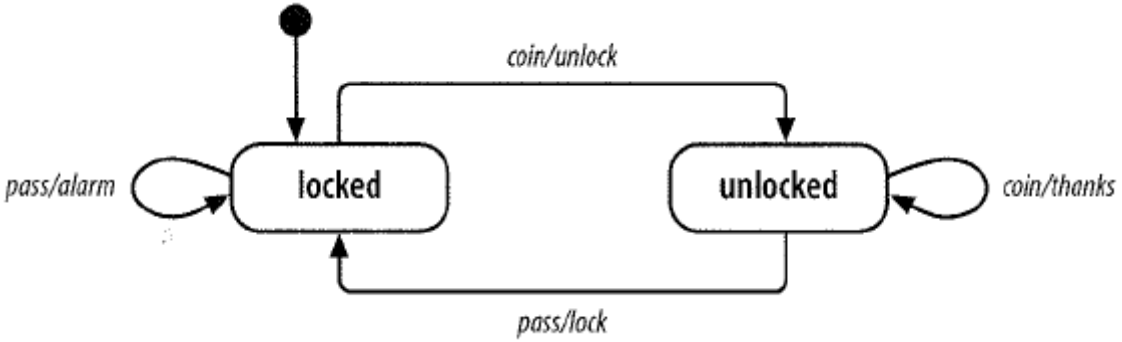
Games Framework

- Bug Runner
  - see also: Java-code



Bug Runner Class Diagram

State Chart ➔ State Transition Table

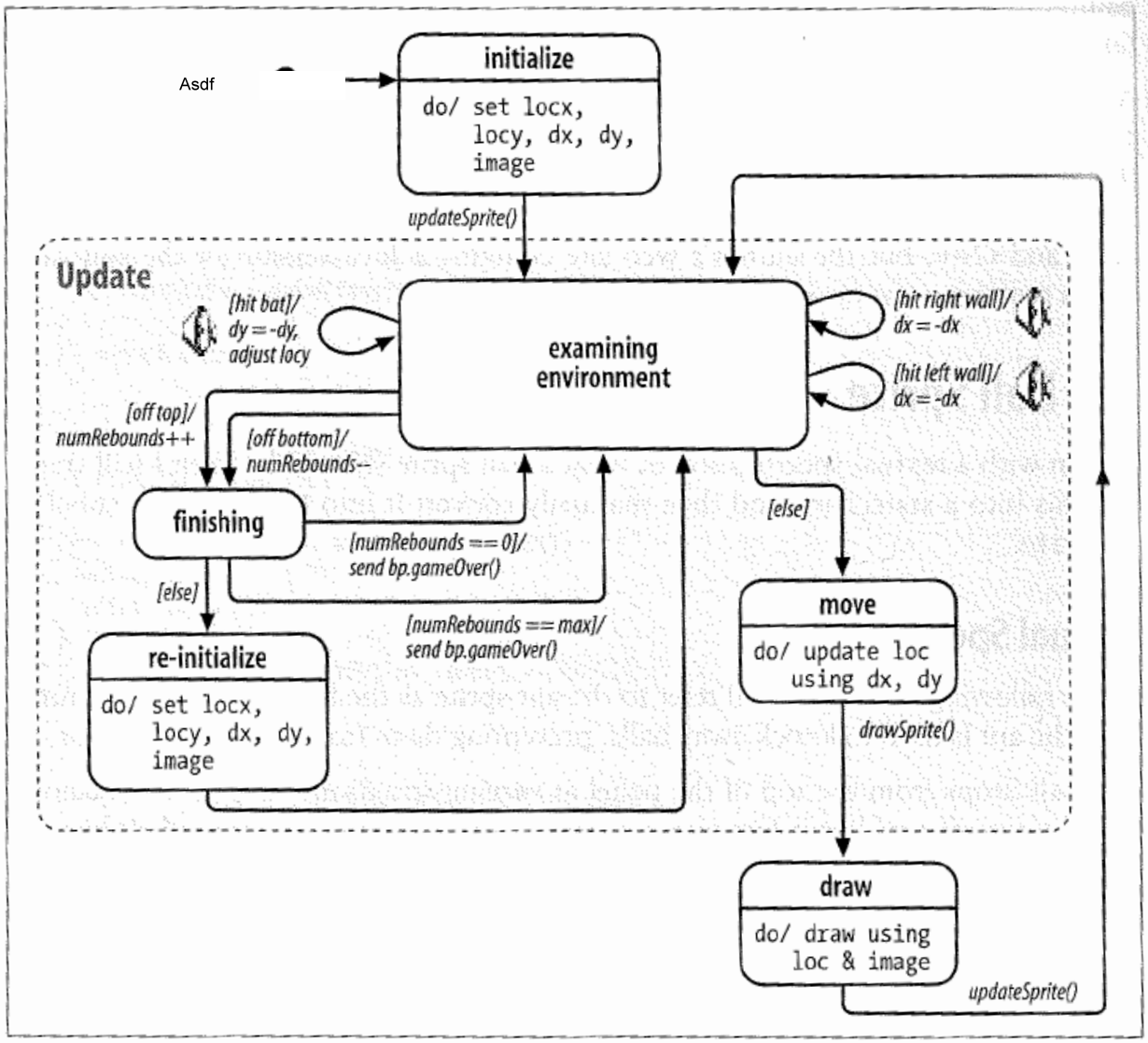


Current State	Event	Action	New State
Locked	Coin	Unlock	Unlocked
Locked	Pass	Alarm	Locked
Unlocked	Coin	Thanks	Unlocked
Unlocked	Pass	Lock	Locked

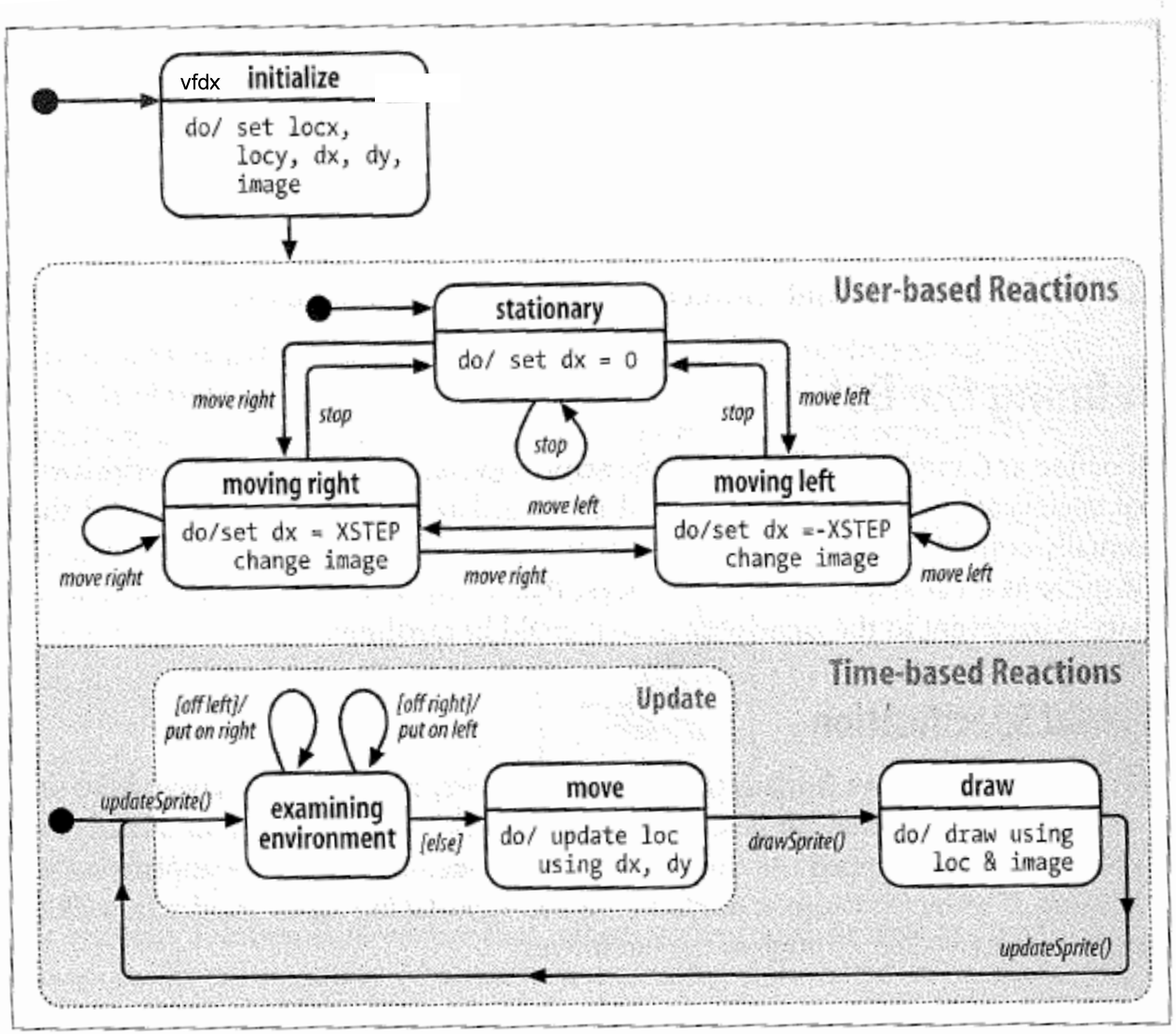
Transition Table ➔ Java Code

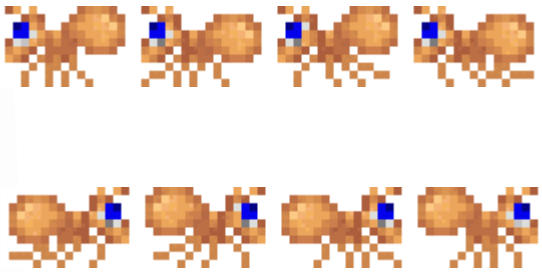

```
public static void main(String args[])
{
  int currentState = LOCKED;
  int event;
  while (true) {
    event = /* get the next event */;
    currentState = makeTransition(currentState, event);
  }
} // end of main()
```

```
private static int makeTransition(int state, int event)
// a translation of Table 1
{
  if ((state == LOCKED) && (event == COIN)) {
    unlock();
    return UNLOCKED;
  }
  else if ((state == LOCKED) && (event == PASS)) {
    alarm();
    return LOCKED;
  }
  else if ((state == UNLOCKED) && (event == COIN)) {
    thanks();
    return UNLOCKED;
  }
  else if ((state == UNLOCKED) && (event == PASS)) {
    lock();
    return LOCKED;
  }
  else {
    System.out.println("Unknown state event");
    System.exit(0);
  }
} // end of makeTransition()
```



Ball Sprite State Diagram





Bug (Player) Sprite State Diagram

JumpingJack - Bricks

http://www.uni-klu.ac.at

- Brick-images are stored in an image-strip
  - /images/

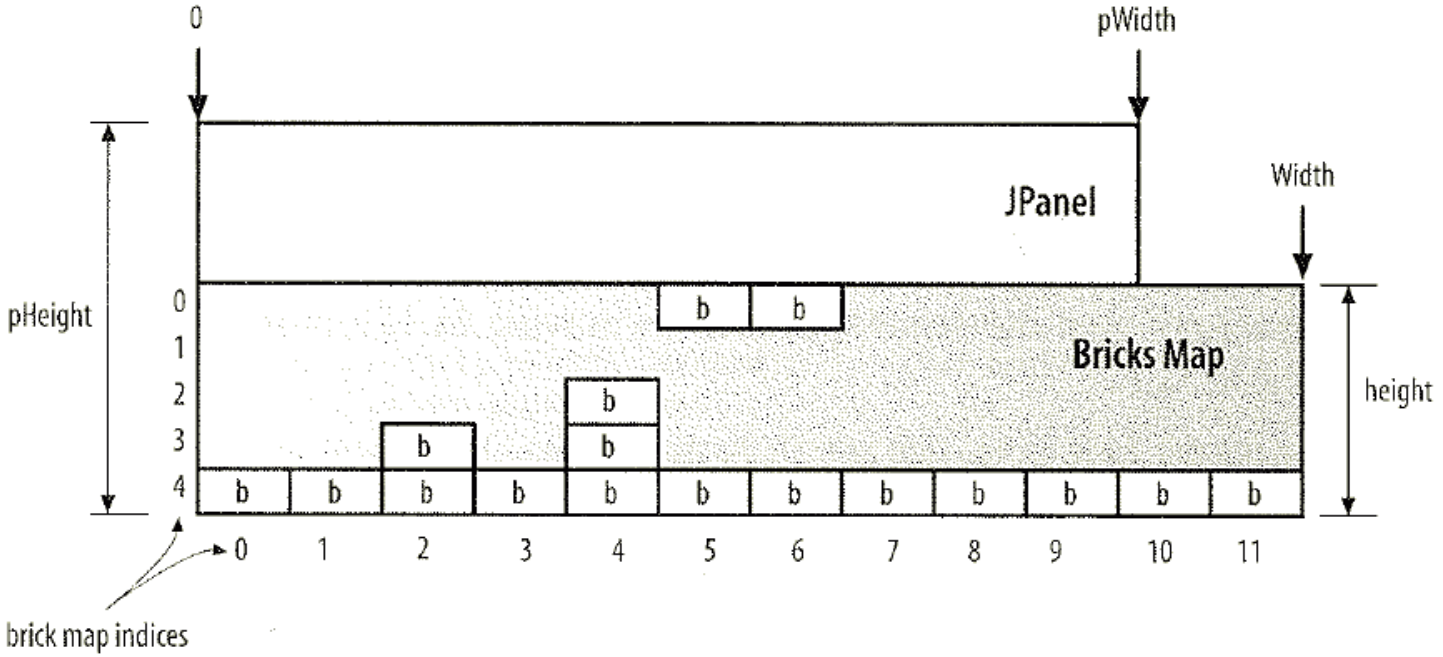
0 1 2 3 4

- Brick-patterns are stored in text-files
  - /images/bricksinfo.txt

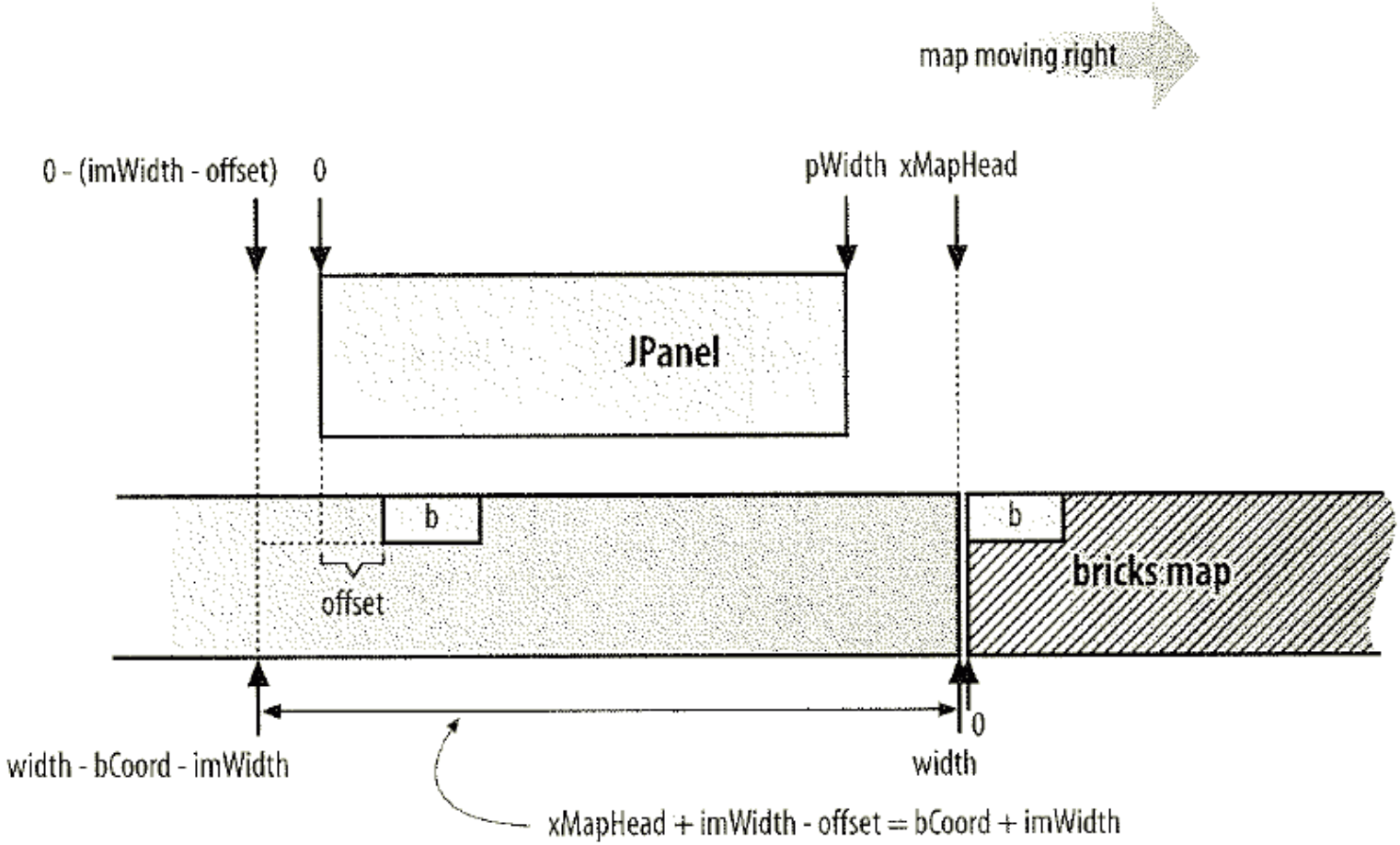
```
                                                   44444
                                               222222222
                                                     111
                                                    2222
                                                   11111
                                                     444
                                                     444
                           22222       444           111
                   1111112222222    23333  2    33    44444444
   00 000111333333000000222222233333  333 2222222223333301
000000000111000000000022200000000033000001111112222222234
```

JumpingJack - Drawing

- Drawing sequence
  - background
    - wrap-around ribbons
    - parallax scrolling)
  - bricks
  - sprites



JumpingJack – Offset for Bricks



„Intelligence“ in Games

- originally „enemy-intelligence“ was stored in fixed patterns, e.g. Pacman:
  - Blinky – chases the player
  - Pinky – ambushes (roundabout)
  - Inky – random movement, starts chasing when close
  - Clyde – stupid, moves completely randomly

  [remark: combination Blinky-Pinky is deadly]
- usually bosses in „boss-fights“ have fixed movement patterns

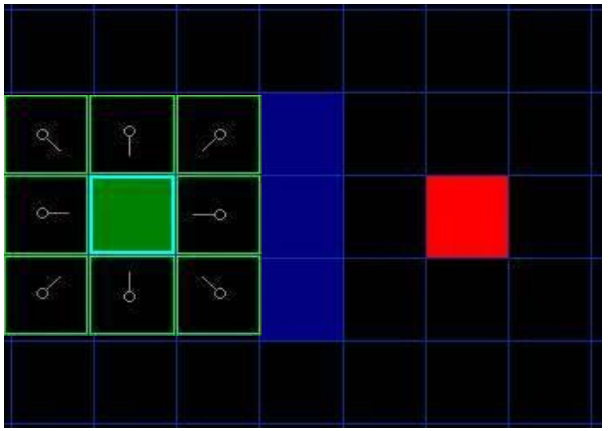
AI in (non-board) Games

- In the 1990s
  - pathfinding problems
  - incomplete information algorithms („fog of war“)
- In the 2000s
  - machine learning algorithms (e.g. neural networks)
  - emergent behavior
    - study and design of complex/multi-agent systems
    - swarm-intelligence (flocking of birds)

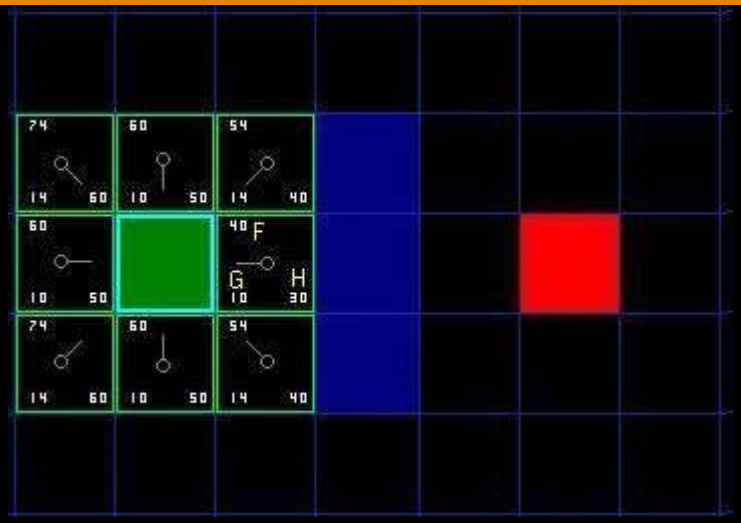
  ➔ enemies without „cheating“ AI

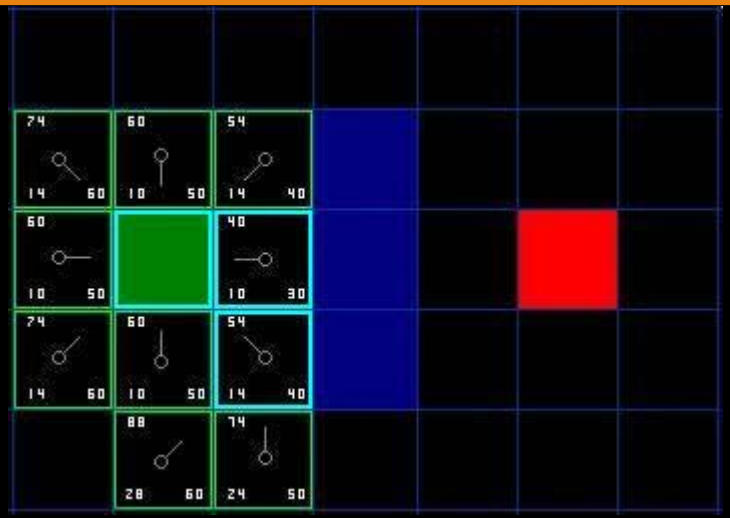
Pathfinding with A*

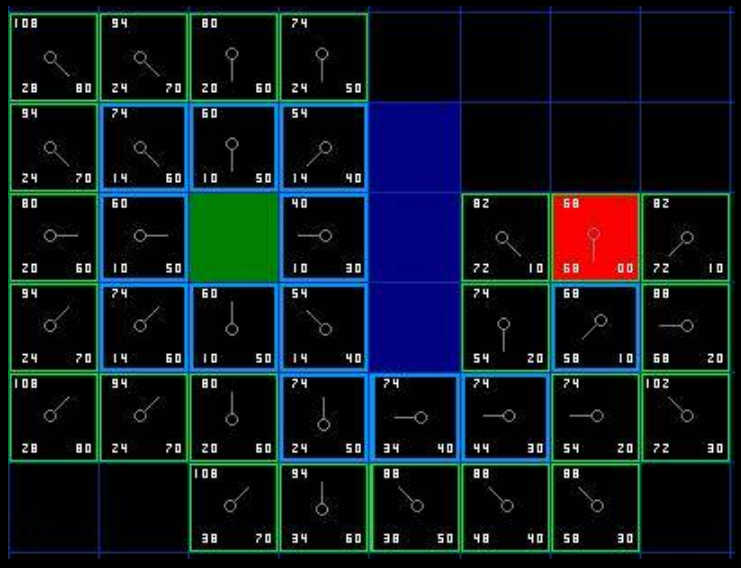
- Given
  - terrain
  - starting position
  - goal position
- Find shortest path
  - span tree
    - calculate new position according to possible movement
    - calculate f = g + h
      - g … cost from starting position (covered distance)
      - h … heuristic: estimated distance to goal
    - span tree from new position
    - stop
      - if shortest path found
      - depth or time limit reached
  - Move to first position on calculated shortest path
- Admissibility Theorem
  - h must be less or equal to the real rest-distance!
  - for our example: h calculated by the manhattan distance

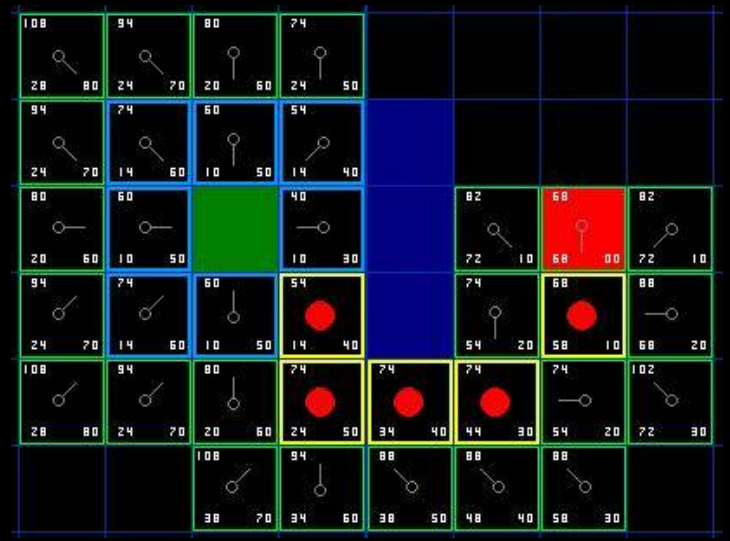


Pathfinding with A*









AI-Thread

- Own thread for AI-calculations
  - PathFinder(**GameState** state) implements Runnable
    - run [implements loop]
      - calculate path(s)
      - store path(s) information in game state
      - sleep(n), value of n depends on
        - » wanted behavior
        - » CPU utilization
- Attention:
  - synchronization!

Some thoughts on Tactial AI

http://www.uni-klu.ac.at

- Tactical movement
  - avoid cones of fire
  - do not move in „lines of sight“
  - seek cover: move to cover locations (predefined)

  ➔ shortest path can be modified quite easily
- Choke point analysis
  - choke-point = points where a player must go through
  - use navigation graph to find choke points
    - graph expresses connections between „rooms“ and „corridors“
  - go to choke points
  - wait for player (ambush)
- Influence Maps
  - dominance of teams per area
  - needed for tactical reasoning
- Commander AI
  - determine kinds of units available
  - select pre-defined tactics based on available information (e.g. „tank-rush“ in C&C)
  - produce needed units
  - command units to hot spots, choke points, under-dominated areas

The game project

Schedule:

- Schedule
  - Deadline: 20.06.2008
- Organizational Issues
  - form groups (3 students)

Source: http://www.cs.wisc.edu/graphics/Courses/679-s2007/Main/GameDesign

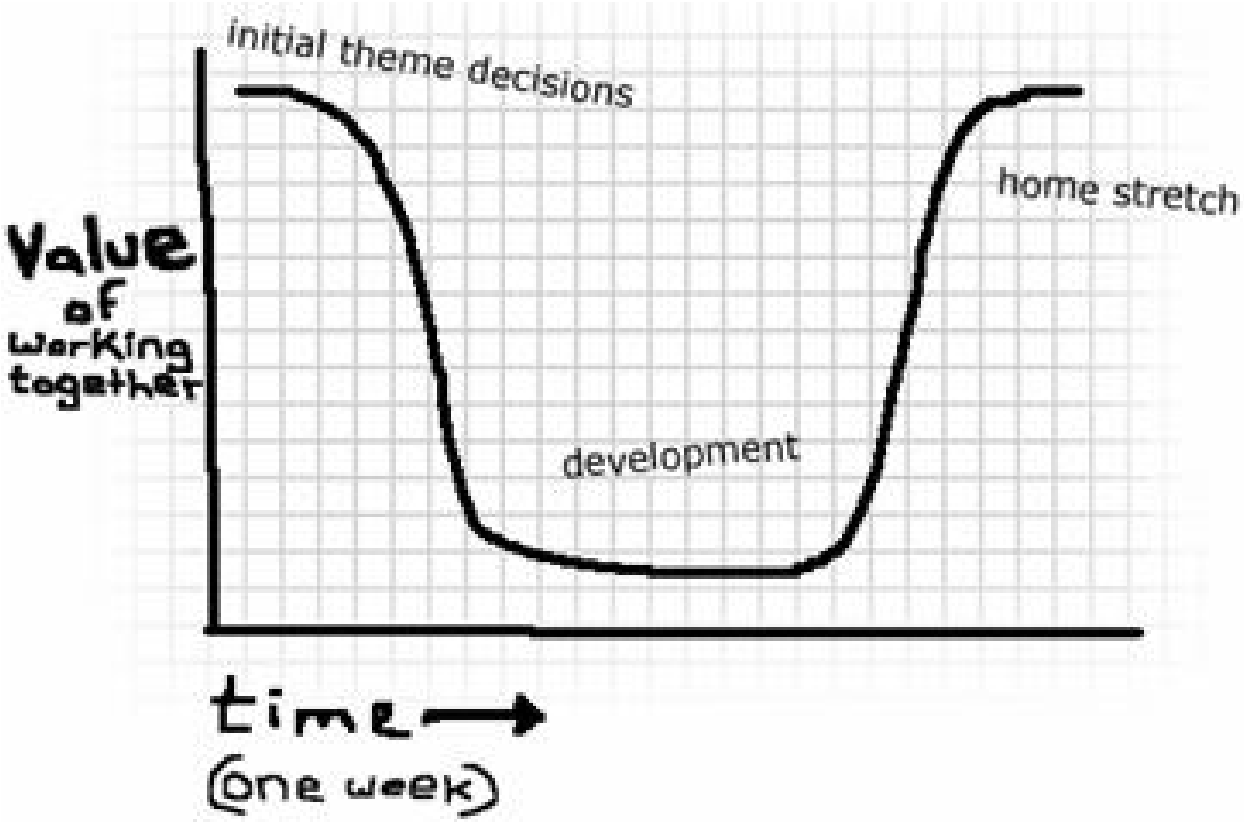
It‘s still just software

- try to finish the brainstorming & design phase before you start programming
- late design changes will cost a lot of time
- differentiate between must-have and nice-to-have
- try to split it up in packages
  - allows to develop in a distributed fashion
  - synchronization
    - define interfaces
    - create mock-ups

Prototyping Approach ...

- gather a kickass team and a good advisor
- gather concept art & music to create an emotional target
- constrain creativity (you'll want it even more)
- complexity is not necessary for fun
- create a sense of ownership
- mindset is as important as talent
- enforce short development cycles
- build towards a well defined game goal
- develop in parallel
- if you can get away with it, fake it

Team Work



[http://www.gamasutra.com/features/20051026/gabler_pfv.htm]

Plan carefully

- planning
  - try to make a schedule (I know: it‘s hard to estimate …)
  - be fair to your team-mates and hold the schedule!
  - a lot of time is usually consumed by testing, debugging and refinement
- be aware of other engagements and commitments
  - there are still other courses & exams
  - summer is approaching
  - soccer fans: EM is approaching
  - do not underestimate the time you will have to invest on new tools (graphics, sound, etc.)

Last but not least

- be motivated, but not over-motivated
  - hard deadline!
- be aware of your own limits
  - a C&C-clone would certainly be a great game, but ….