PAPER CODE NO.	EXAMINER : Boris Konev
COMP222	DEPARTMENT : Computer Science Tel. No. 0151 795 4260

UNIVERSITY OF LIVERPOOL

Second Semester Examinations 2016/17

Principles of Computer Game Design and Implementation

TIME ALLOWED : Two Hours

INSTRUCTIONS TO CANDIDATES

Answer FOUR questions.

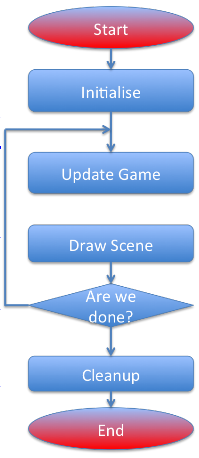
If you attempt to answer more questions than the required number of questions (in any section), the marks awarded for the excess questions answered will be discarded (starting with your lowest mark).

UNIVERSITY OF LIVERPOOL

Question 1

A. Describe the code structure of a modern computer game. Your answer should mention *game-specific* code, *game engine*, and *in-house tools*. You should also cover a typical game architecture to include *initialisation*, *main game loop* and *cleanup*. Give a diagrammatic representation of a typical game architecture. **7 marks**

Most games make a distinction between game-specific code and game-engine code. Game-specific code is, as the name implies, tailored to the current game being developed. It involves the implementation of specific parts of the game domain itself, such as the behaviour of zombies or spaceships, tactical reasoning for a set of units, or the logic for a front-end screen. This code is not intended to be generically reused in any other game in the future other than possibly direct sequels.
Game-engine code is the foundation on top of which the game-specific code is built. It has no concept of the specifics of the game being developed, and deals with generic concepts that apply to any project: rendering, message passing, sound playback, collision detection, or network communication.
In-house development tools are created to support artists and developers in creation of worlds and content.
Initialisation and shutdown of different systems and phases of the game is a very important step, yet it is often overlooked. Without a clean and robust way of initialising and shutting down different parts of the game, it becomes very difficult and error prone to switch between levels, to toggle back and forth between the game and the front end, or to even run the game for a few hours without crashes or slowdowns.
Main Game Loop: At their heart, games are driven by a game loop that performs a series of tasks every frame. By doing those tasks every frame, we put together the illusion of an animated, living world. Sometimes, games will have separate loops for the front end and the game itself, since the front end usually involves a smaller subset of tasks than the game. Other times, games are organised around a unified main loop.



A description of the code structure gives 3 marks, 1 mark for the picture; description so initialisation, game loop and cleanup is 1 mark each.

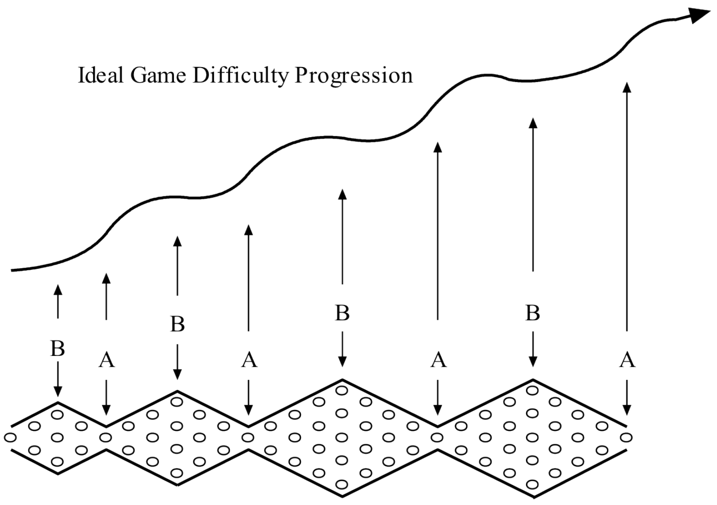
B. Classify every one of the following

- rendering,
- behaviour specific to zombies,
- message passing,
- sound playback

as a part of game engine code or game-specific code. **2 marks**

Rendering, message passing and sound playback belong go game engine. Behaviour of zombies and level implementation are a part of game-specific code.

C. In your opinion, why do game developers often design games so that periods of increased difficulty follow more relaxed periods, as shown in the diagram below?



In your answer discuss

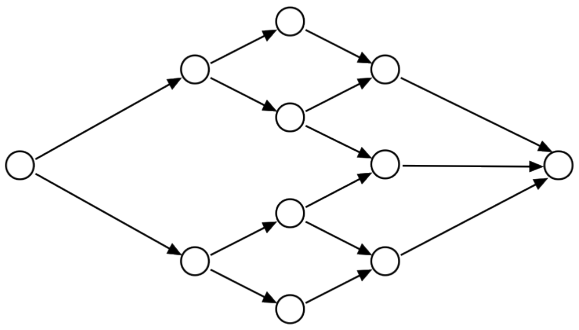
- the link between the difficulty progression and the definition of a computer game as a sequence of meaningful choices made by the player in pursuit of a clear and compelling goal;
- how this form of difficulty progression fits the classical game structure;
- what points A and B in the diagram correspond to.

7 marks

A computer game can be seen as a sequence of meaningful choices made by the player in pursuit of a clear and compelling goal because player

- must have choice, or it is not interactive;
- must be a series of choices or it is too simple to be a game;
- must have a goal or it is a software toy.

Graphically, the classical game structure can be represented as



Starts with a single choice, widens to many choices, returns to a single choice. The narrow decision points (A) in the convexities where players' choices are limited can be set to correspond to the difficult parts of the game. Designers have come up with many game mechanisms to make this happen, though boss monsters or climactic battles. The wider points (B) where there are many alternatives give the player more discretion and typically difficulty increases more slowly, or even decreases a bit to provide a break for the player and let him or her rehearse new skills.
The justification of a definition points is 1 mark each (3 max) A diagram is 2 marks. The argument why the structure works is 2 marks.

D. Name at least two advantages and at least two disadvantages of using a scripting language such as Python or Lua for programming game-specific code. **4 marks**

Game-specific code involves the implementation of specific parts of the game domain itself, such as the behaviour of zombies or spaceships, tactical reasoning for a set of units, or the logic for a front-end screen. This code is not intended to be generically reused in any other game in the future other than possibly direct sequels.
Advantages of using a scripting language:

- Ease and speed of development
- Short iteration time
- Code becomes a game asset
- Offer additional features and are customizable Can be mastered by artists / designers

Disadvantages of using a scripting language:

- Slow performance
- Limited tool support
- Dynamic typing makes it difficult to catch errors
- Awkward interface with the rest of the game
- Difficult to implement well

Every advantage / disadvantage is 1 mark.

E. Games are driven by a game loop that performs a series of tasks every frame. Traditionally, games only featured one game loop. Some game architectures suggest running multiple game loops in different threads. What is the reason for doing this? Name at least one advantage and one disadvantage of such an approach to multithreading. **5 marks**

This architecture allows the developers to run different game subsistems in different update loops, e.g. Heads-up Display, AI, Rendering, Physics, etc all run in different threads.
Advantages:

- Better use of hardware resources;
- Clear separation between subsystems;
- Developers can run different subsystems at different speeds.

Disadvantages

- The cost of switching context on older architecture (where multithreading is software not hardware driven) – less important these days;
- Game subsystems are tightly coupled and making the execution thread safe might be tricky;
- Most of the time the threads will be waiting – e.g. the physics simulation thread will wait until new updates come from the user.

Every advantage / disadvantage is 1 mark. A comparison of a single loop / multiple loops is 2 marks. For 1 extra mark: a much better approach is to split the load into small independent units and run them on the available CPUs.

Question 2

A. Let $\mathbf{V} = (3, 2, 1)$ and $\mathbf{W} = (4, 2, -6)$ be 3D-vectors. Compute (and show your working)

(a) $\mathbf{V} + \mathbf{W}$ **1 marks**

$$\mathbf{V} + \mathbf{W} = (3 + 4, 2 + 2, 1 - 6) = (7, 4, -5)$$

(b) $2\mathbf{V}$ **1 marks**

$$2\mathbf{V} = (6, 4, 2)$$

(c) $\mathbf{V} - 2\mathbf{V}$ **1 marks**

$$\mathbf{V} - 2\mathbf{V} = -\mathbf{V} = (-3, -2, -1)$$

(d) $\mathbf{V} \cdot \mathbf{W}$ **1 marks**

$$(3, 2, 1) \cdot (4, 2, -6) = 3 \cdot 4 + 2 \cdot 2 - 1 \cdot 6 = 12 + 4 - 6 = 10$$

(e) $|\mathbf{V}|$ **1 marks**

$$|\mathbf{V}| = \sqrt{3^2 + 2^2 + 1^2} = \sqrt{14}$$

(f) $\mathsf{proj}_{\mathbf{V}}\mathbf{W}$ **1 marks**

$$\mathsf{proj}_{\mathbf{V}}\mathbf{W} = \frac{\mathbf{W} \cdot \mathbf{V}}{\|\mathbf{V}\|^2}\mathbf{V},$$

so

$$\mathsf{proj}_{\mathbf{V}}\mathbf{W} = \frac{10}{14}(3, 2, 1) = \left(\frac{15}{7}, \frac{10}{7}, \frac{5}{7}\right)$$

(g) $\mathbf{V} \times \mathbf{W}$ **2 marks**

$$(3, 2, 1) \times (4, 2, -6) = (-2*6-1*2, 1*4+3*6, 3*2-2*4) = (-14, 22, -2)$$

B. Recall that the 2D rotation matrix representing the counter-clockwise rotation about the origin by an angle θ is as follows

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Give the 3D rotation matrix representing the counter-clockwise rotation about the Z-axis through an angle α combined with counter-clockwise rotation about the X-axis through an angle β. **8 marks**

Rotation about the Z-axis is given by

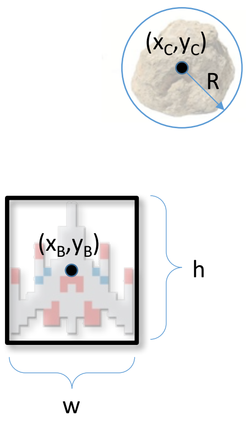
$$M_Z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation about the X-axis is given by

$$M_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix}$$

Their combination is given by the matrix produce $M_Z \times M_X$.

C. Assume you are implementing a simple 2D space shooter arcade game. In this game a user is in control of a spaceship, which is approximated by an axis-aligned bounding box. You need to implement collision detection with obstacles that are approximated as circles.

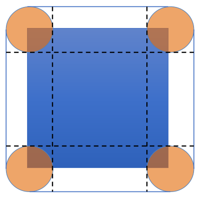


Given the dimensions h and w of the box, coordinates (x_B, y_B) of its centre, coordinates (x_C, y_C) of the circle and the circle radius R

- Draw a diagram representing when a collision happens; **2 marks**
- Clearly identify the cases you need to consider; **2 marks**
- Sketch a method which determines whether the box overlaps the circle. **5 marks**

This problem has not been seen by the students.

- Diagram:



- Cases (taken from http://stackoverflow.com/questions/21089959/detecting-collision-of-rectangle-with-circle):

Step 1: Find the vertical and horizontal (distX/distY) distances between the circles center and the rectangles center
var distX = Math.abs(circle.x - rect.x-rect.w/2);
var distY = Math.abs(circle.y - rect.y-rect.h/2);

Step 2: If the distance is greater than halfCircle + halfRect, then they are too far apart to be colliding
if (distX > (rect.w/2 + circle.r)) { return false; }
if (distY > (rect.h/2 + circle.r)) { return false; }

Step 3: If the distance is less than halfRect then they are definitely colliding
if (distX <= (rect.w/2)) { return true; }
if (distY <= (rect.h/2)) { return true; }

Step 4: Test for collision at rect corner.
Think of a line from the rect center to any rect corner. Now extend that line by the radius of the circle. If the circles center is on that line they are colliding at exactly that rect corner Using Pythagoras formula to compare the distance between circle and rect centers.
var dx=distX-rect.w/2;
var dy=distY-rect.h/2;
return (dx*dx+dy*dy<=(circle.r*circle.r));

-

```
// return true if the rectangle and circle are colliding
function RectCircleColliding(circle,rect){
    var distX = Math.abs(circle.x - rect.x-rect.w/2);

    var distY = Math.abs(circle.y - rect.y-rect.h/2);

    if (distX > (rect.w/2 + circle.r)) { return false; }

    if (distY > (rect.h/2 + circle.r)) { return false; }

    if (distX <= (rect.w/2)) { return true; }
    if (distY <= (rect.h/2)) { return true; }

    var dx=distX-rect.w/2;
    var dy=distY-rect.h/2;
    return (dx*dx+dy*dy<=(circle.r*circle.r));
}
```

Question 3

A. Modern computer games commonly use scene graphs to represent a graphical scene. Name at least three advantages of this form of data representation as compared with unstructured collections of geometries, light sources, textures, etc. **3 marks**

Some advantages:

- Scene graphs provide an intuitive way to manage large amounts of geometric and rendering data
- The data needed for rendering, which is associated with the scene graph nodes, can be kept separate from the rendering code.
- Hierarchical animated models are easier to deal with
- View frustum culling can be supported by using bounding volumes at the nodes.

Every advantage contributes 1 mark (up to 3).

B. Describe the role of a renderer in a game engine. **2 marks**

A renderer

- Transforms geometry from world space to screen space
- Eliminates hidden objects
- Draws the transformed scene

C. Collision detection based on overlap testing may lead to penetration and tunnelling. Explain what is meant by these terms. Name at least two methods to avoid penetration and tunnelling. **4 marks**

The idea of overlap testing is that at every simulation step, each pair of objects will be tested to determine if they overlap with each other. If two objects overlap, they are in collision. This is known as a discrete test since only one particular point in time is being tested.
The biggest disadvantage of an overlap testing is that it handles poorly objects travelling fast, so object can get inside each other (penetration) or travel through each other (tunnelling).
For overlap testing to always work, the speed of the fastest object in the scene multiplied by the time step must be less than the size of the smallest collidable object in the scene. This implies a design constraint on the game to keep objects from moving too fast relative to the size of other objects.
Some methods to avoid penetration and tunnelling:

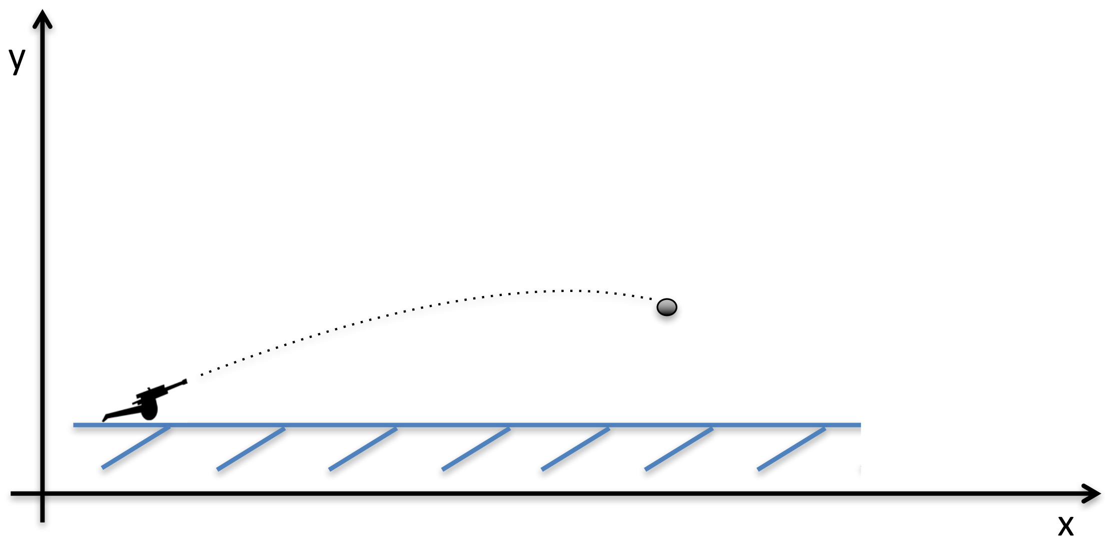
- smaller timesteps
- extruding the object along the motion
- ray cast to the new position

Explanations are worth 2 marks (1 mark per method). 2 marks for coming up with a solution.

D. Is it possible to combine animation-based collision response with a physics-based collision response in a game? If yes, give an example; if not provide a justification. **4 marks**

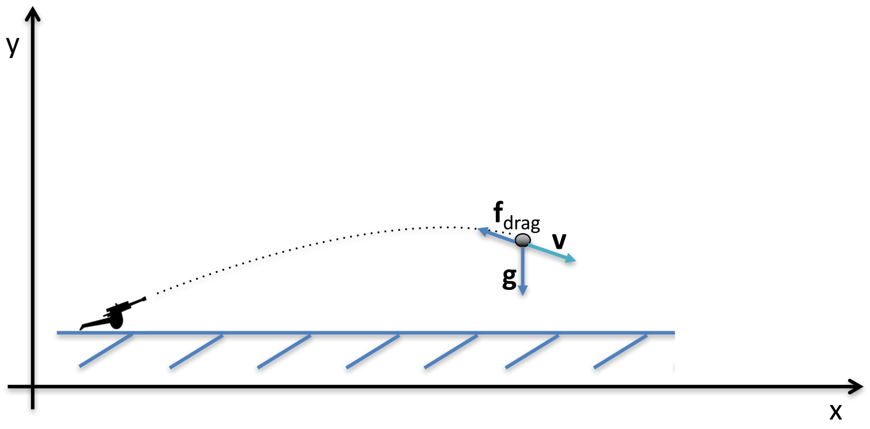
It is possible. Popular AAA games use a clever mix of physics, animation and prerendered graphics to give you the illusion of a real, physical world. For example, a building breaking into 4-8 parts after an explosion. The pieces most likely fly on predefined (so called kinematic) paths and are only replaced by dynamic Spatials after they touch the ground.

E. Consider a 2D game, in which a gun fires a cannonball. As part of the gameplay, you are modelling the effect of the air resistance on the cannonball. The mass of the cannonball is 50kg. The initial speed vector for the cannonball is $(100, 50)$.



Assuming the linear model of drag,

(a) give a graphical representation of all the forces acting on the cannonball as it flies through the air; **2 marks**



Where **v** is the speed vector and **g** is gravity pull

(b) describe the discrete motion of the cannonball as a sequence of its positions using Euler steps; **5 marks**

To use Euler steps, we need to upate the forces, acceleration, velocity and position of the cannonball at every frame as follows.

$$\begin{aligned}
\mathbf{F}_{i+1} &= -b \cdot \mathbf{V}_i \\
\mathbf{a}_{i+1} &= \mathbf{g} + \frac{1}{m} \cdot \mathbf{F}_{i+1} \\
\mathbf{V}_{i+1} &= \mathbf{V}_i + tpf \cdot \mathbf{a}_{i+1} \\
\mathbf{P}_{i+1} &= \mathbf{P}_i + tpf \cdot \mathbf{V}_{i+1}
\end{aligned}$$

(c) sketch the `simpleUpdate()` method that implements the described motion in jMonkeyEngine. You are not required to write finished working code, but you must clearly convey the idea. **5 marks**

```
Vector3f force,acceleration;
Vector3f velocity = new Vector3f(100,50,0);
protected void simpleUpdate() {
     force = velocity.mult(-b);
     acceleration = gravity.add(force.divide(m));
     velocity = velocity.add(acceleration.mult(tpf));
     s.setLocalTranslation(s.getLocalTranslation().
              add(velocity.mult(tpf)));
}
```

Question 4

A. Poor collision detection can lead to artefacts in computer games. Name at least two undesirable implications of poor collision detection in computer games. **4 marks**

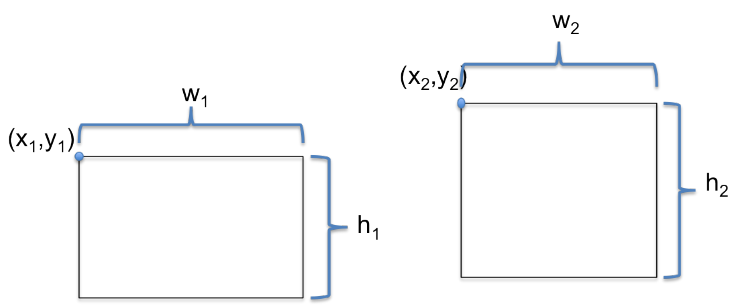
- Players/objects falling through the floor;
- Projectiles passing through targets;
- Players getting where they should not get;
- Players missing a trigger boundary.

Every one contributes 2 marks (up to 4).

B. Overlap testing in computer games is often approximated with the help of *bounding volumes*: a real shape is being embedded into a simplified geometry, and if two bounding volumes do not overlap, one does not perform an (expensive) triangle-level overlap test.

(a) Simple bounding volume shapes include Axis Aliened Bounding Boxes (AABBs) and Oriented Bounding Boxes (OBBs). What are the advantages of OBBs over AABBs? Are there any significant disadvantages? Your answer should contain definitions of OBBs and AABBs. **4 marks**
OBBs stands for oriented bounding boxes; AABBs stands for axis aligned bounding boxes.
OBBs fit the real geometry tighter. The main disadvantage is that it is harder to check if the bounding volumes overlap; however, this disadvantage is outweighed by better collision detection offered by OBBs.
Explaining abbreviations - 1 mark each. Explanations of what they are 1 mark per definition.

(b) Sketch a method which, given the coordinates of *upper left* corners of two 2-dimensional axis-aligned boxes (x_1, y_1) and (x_2, y_2) and their widths w_1, w_2 and heights h_1, h_2, respectively, determines whether these boxes intersect.



7 marks

There is an elegant solution to this problem based on the check of when boxes *do not* overlap.

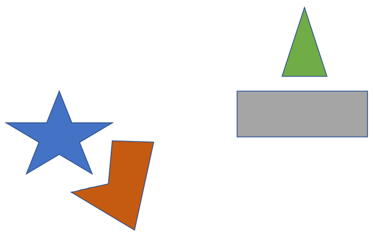
```
if((x1 + w1 < x2) || (x2 + w2 < y2) ||
    (y1 + h1 > y2) || (y2 + w2 > y1)) {
        return false;
}
else {
        return true;
}
```

C. Recall that bounding volume hierarchies are used to better support collision detection.

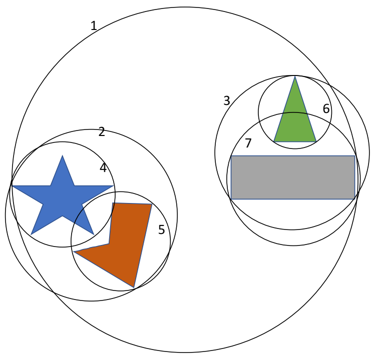
(a) Explain how bounding volume hierarchies are used in collision detection. **2 marks**

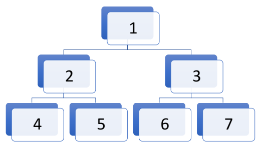
Collision detection in computer games is often approximated with the help of *bounding volumes* – a real shape is being embedded into a simplified geometry, and if two bounding volumes do not overlap, one does not perform an (expensive) triangle-level overlap test.
Although the tests themselves have been simplified, the same number of pairwise tests are still being performed.
A BVH a hierarchical structure, in which the root node completely encapsulates the object; Children give a "tighter fit" for the shape; and Recursive / iterative algorithms are used to construct and navigate BVHs.

(b) Sketch a bounding volume hierarchy based on **bounding spheres** for the shape given below and use it as an example to illustrate your answer.

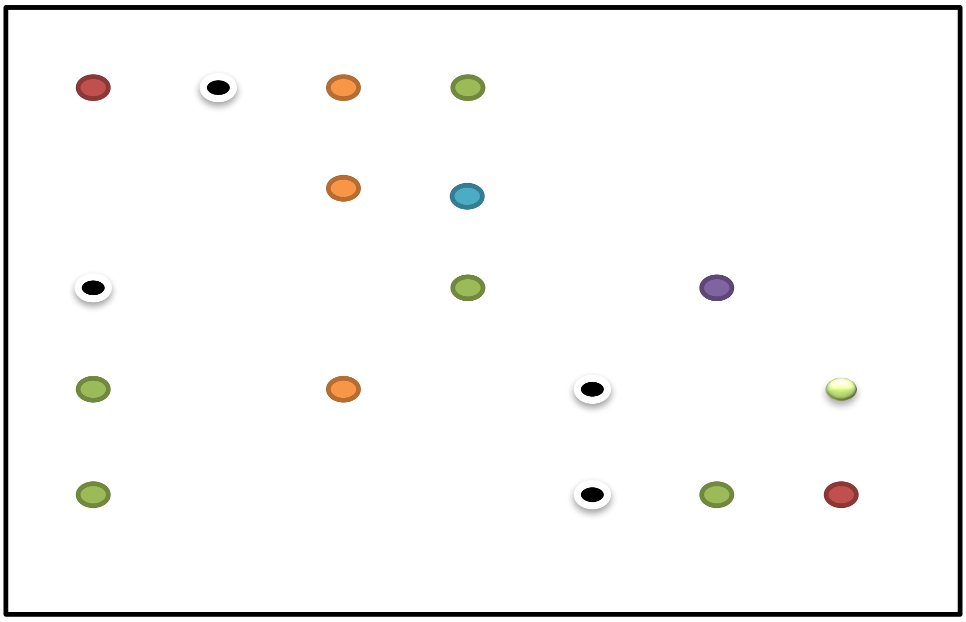


5 marks


1
3
6
2
7
4
5


1
2
3
4
5
6
7

D. In your opinion, what data structure is most suitable to reduce the number of pairwise collision detection tests in the scene shown below? Explain your reasoning.

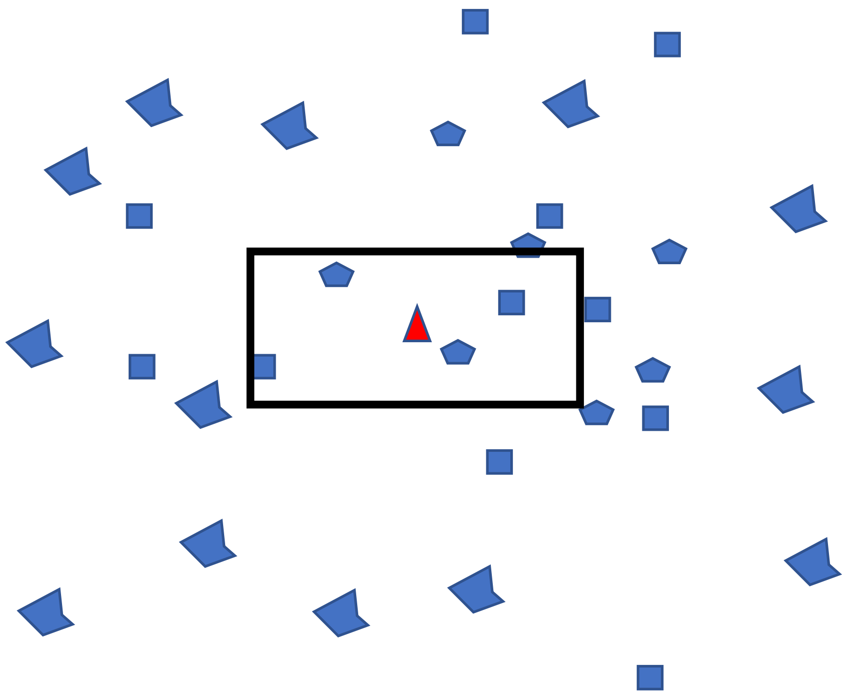


3 marks

Objects are similar size and but not uniformly distributed. Thus either an implicit grid (e.g. spatial hash) or a quad tree, a k-d tree of a BSP tree are best suited.

Question 5

A. You are implementing a 2D asteroids field video game targeting a very weak computational platform. In your game, you want to have in excess of 1000 asteroids floating freely in the space. You want to model how asteroids collide and, on impact, break up into smaller ones. In the centre of your viewport (shown as a frame in the image below) you have a spaceship (shown as a triangle). As the spaceship moves in the space, the viewport follows.



Having implemented accurate collision detection based on the low-level overlap test, you discover that running it on every pair of asteroids leads to very poor performance. What techniques can you suggest to speed-up the game? **10 marks**

This is is an open-ended problem solving question. Students can suggest own techniques. Some techniques mentioned in the lectures:

100% optimisation Only run simulation on the visible part. Asteroids will be spawn in the near proximity of the viewport.

Spatial data structure Use a spatial data structure such as a quad-tree to determine which asteroids are close and have a chance of colliding

Simplified geometries Use simplified geometries. Bounding spheres (circles) seem to be a good choice here.

B. In computer game AI one can often identify two actors: a virtual player and a game agent.

- Define what they are and what role they play in computer games.
- Give an example of both.
- What is the difference between them?

- Give examples of when a computer game has a virtual player but no game agents and when a computer game has game agents but no virtual player.

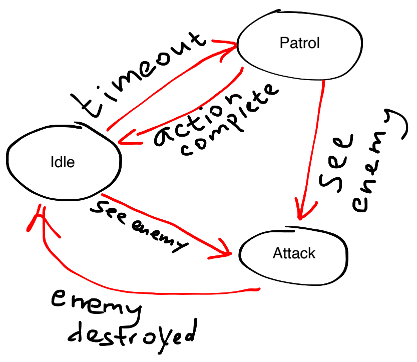
- How do a virtual player and game agents collaborate? Give an example.

4 marks

Game agents are autonomous entities that observe and act upon an environment. They often are associated with game characters (enemies, companions, computer car drivers etc.). Early agents did not show much of intelligent behaviour, often their choices were random. In modern computer games they can learn and react to the environment in an intelligent way.
A virtual player takes place of a human opponent in a game. The virtual player performs the same operations as the human player. The intelligence of the virtual player is perceived through the moves it makes and the results of choices. For example, a chess player is a virtual player.
Many first-person shooters have game agents (enemies) but no virtual player. Chess has a virtual player but no game agents.
In real-time strategies, the computer-controlled side is a virtual player (thus, there might be more than one virtual player in a game), while individual units are game agents, which often can take decisions on their own in order to follow orders.
The definitions (illustrated with examples) of an agent and a virtual player are 1 mark each. Examples of games – 1 mark extra. Collaboration – 1 mark.

C. Consider the following behaviour of a fighter game agent. The agent can be in three possible states: *idle*, *patrol*, or *attack*. In the *idle* state the agent remains motionless, in the *patrol* state the agent moves to the next checkpoint, and in the *attack* state the agent attacks another player. If the agent sees the other player, it goes into the *attack* state; otherwise, from being idle it changes, on a timeout, to the *patrol* state and, once completed the move to the next checkpoint, returns to the *idle* state. If the enemy unit is destroyed, the agent goes from the *attack* state to the *idle* state.

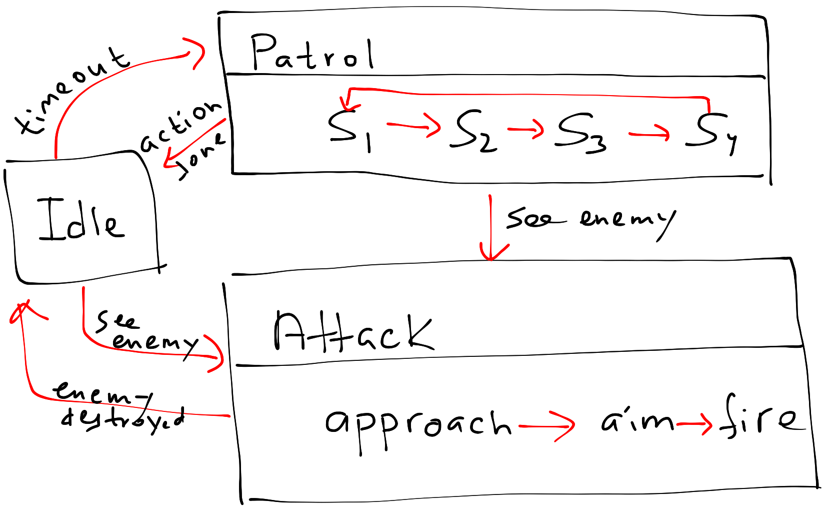
(a) Give a graphical representation of the FSM that represents the agent behaviour. Indicate clearly conditions under which one state changes into another. **5 marks**



(b) Assume now that you want the agent to show more complicated behaviour: in the *patrol* state the agent patrols four stations $S_1,\ldots,S_4$ in the order $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_1 \rightarrow \ldots$ and in the *attack* state the agent goes through three consecutive stages: *approach*, *aim*, *fire*.

In your opinion, what is the best way to accommodate these modifications to the agent behaviour? Give a graphical representation of the new model of agent behaviour. **6 marks**

There are two options how this can be handled. Either to add more states to the FSM, or consider a hierarchical FSM in which the *patrol* state and *attack* state are FSMs as follows.



Question 6

A. Why in computer games is the character motion control routine often considered at two logical levels: steering and pathfinding? Name at least two advantages of such separation. **4 marks**

Steering techniques allow a computer character to navigate from one position into another provided there are no (or few simple) obstacle on the way. Pathfinding is used whenever a computer entity needs to find a way to a goal avoiding obstacles.
Advantages:

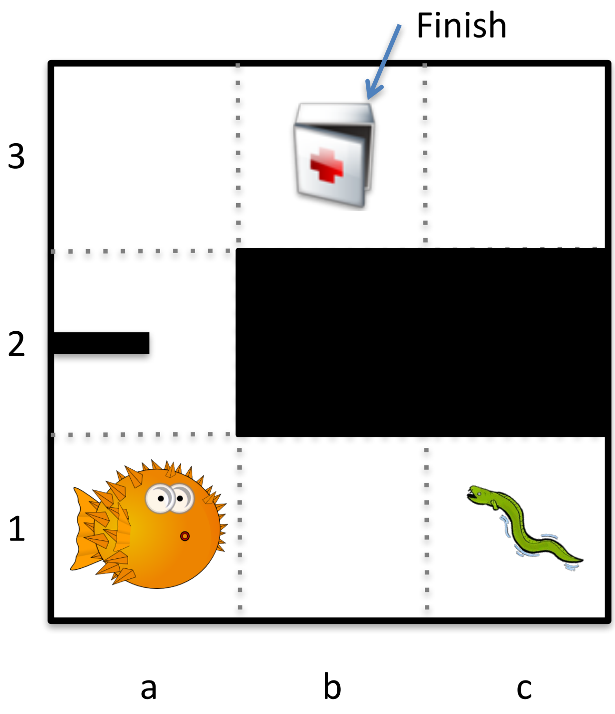
- steering is closer to game engine an often requires integration with the physics engine; pathfinding is closer to the decision taking level. Keeping them separate leads to a cleaner code and better task distribution.
- Pathfinders can be reused in a different kind of game even of another genre.

One mark per advantage.

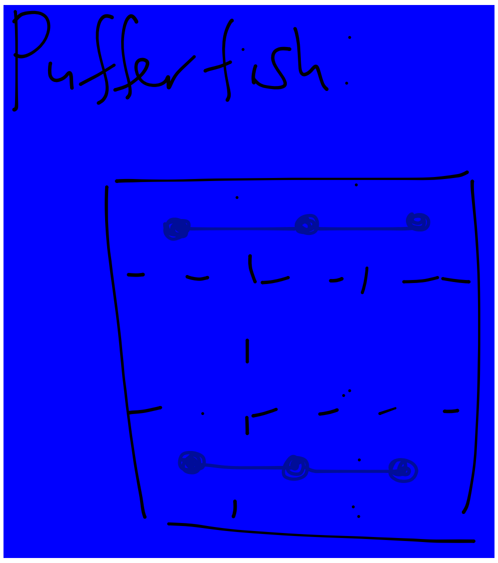
B. What is the difference between static, kinematic and dynamic physics entities in the terminology of computer games physics engine? Give an example of a static, kinematic and dynamic physics entities. **4 marks**

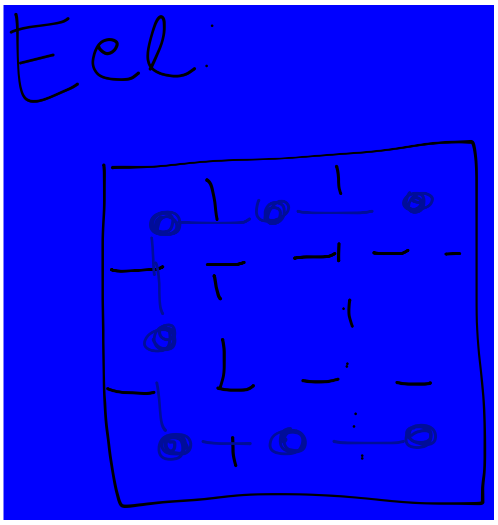
All physics entities can detect collisions and can respond to collision. Static entities never move. An example of those are floors, walls etc. Kinematic entities are controlled by the game logic. They can interact (e.g. 'push') with other physics entities but they are not influenced by other entities. Examples of kinematic entities include airships, elevators, doors, etc. Dynamic entities are controlled by the physics engine and they react to physics interaction with other entities. Examples include rolling balls, movable crates etc.
Definitions – 2 marks; examples – 2 marks.

C. Consider the following floor plan of a 3x3 room. Tiles **2b** and **2c** are impassable. The Pufferfish has filled its stomach with water and so cannot pass through tile **2a** either; however the Eel can.

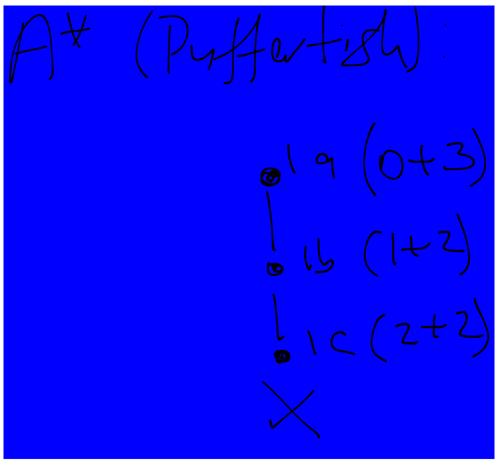


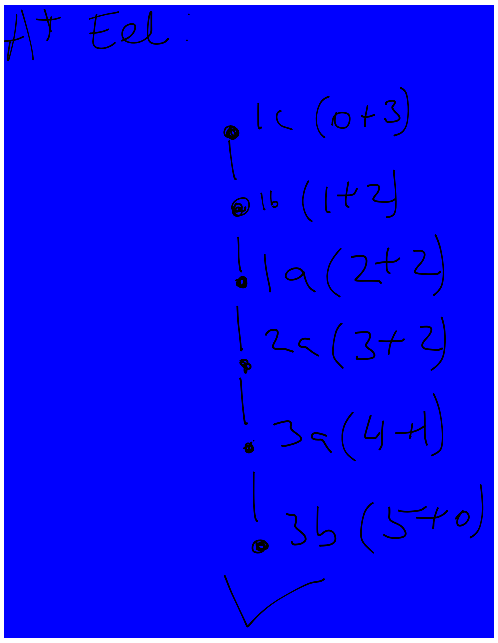
(a) Construct the tile-based pathfinding graphs for both agents (Pufferfish and Eel). **4 marks**





(b) Using the Manhattan block distance between tiles as a heuristic, which tiles and in which order the A* algorithm will explore for both agents when trying to find a path from their current position to the target tile **3b** (we assume that both agents can occupy the same tile)? Illustrate the work of the A* algorithm with a diagram. For every node of the diagram indicate clearly the cost so far and the estimated cost to the goal. **5 marks**





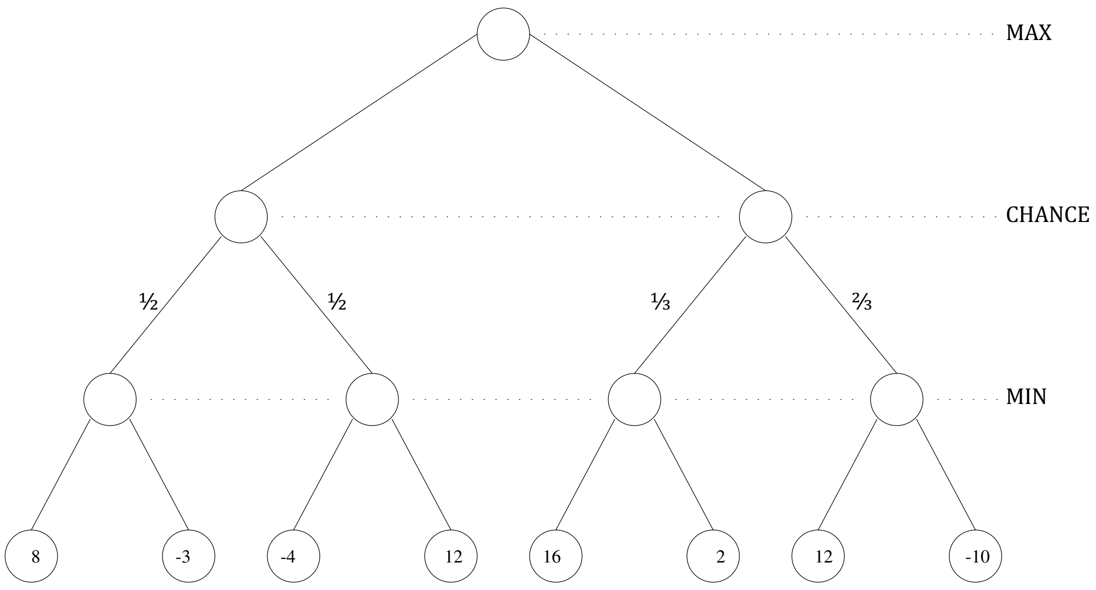
(c) Which techniques can be used to speed up finding that there is no path for the Pufferfish? Show how the technique of your choice works on this example. **5 marks**

Zones and zone equivalence arrays can be used. If the start tile and the end tile belong to the same zone, a path exists. If they don't belong to the same zone and the zone equivalence arrays for the agent does not equate them, there is no path.
With 0 standing for impassable tile, we have the following zone array:

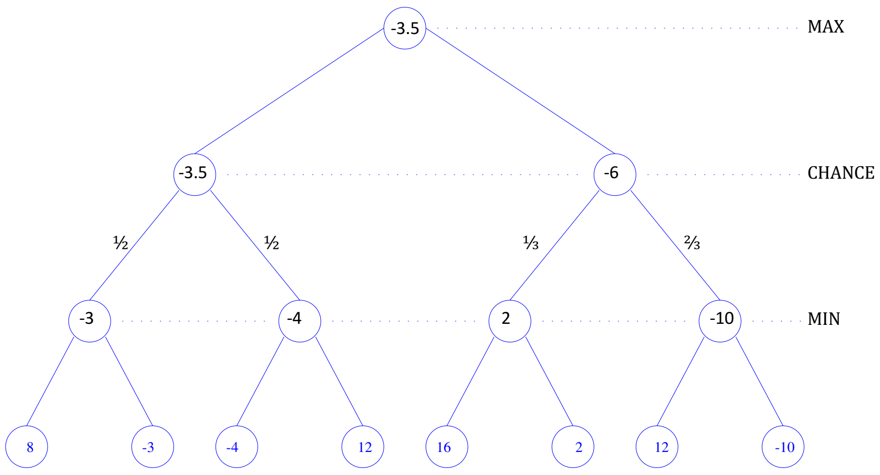
$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \\ 3 & 3 & 3 \end{bmatrix}$$

For Pufferfish, the zone equivalence array is as follows: $ZE_P = [0, 1, 2, 3]$, that is all zones are different. Then the zone of tile **1a** is 3, the zone of **3b** is 1, $ZE_P[1] \neq ZE_P[3]$ so we conclude there is no path.
For Eel, the zone equivalence array is $ZE_E = [0, 1, 1, 1]$ so that zones 1, 2 and 3 are equivalent. As $ZE_E[1] = ZE_E[3]$ we conclude there exists a path for Eel.

D. Consider the following chance game tree. Perform the ExpectiMiniMax algorithm on this tree and compute the ExpectiMiniMax value of the root node.



3 marks



Root node ExpectiMiniMax value: -3.5.