# Games

In this assignment, you should write a game using pygame. You are given a base file ('snake.py' or/and 'flappy_bird.py') that contains starting code: simple update-render game loop discussed during lectures. Running the code will open an empty window. Your task is to extend the code based on the game rules.

This assignment contains a description of 2 games: Flappy Bird and Snake. Each game has a basic structure which is mandatory to do, after that you can fill the remaining points with elective extensions.

**IMPORTANT: Each game should be written in the appropriate file. If you implement Snake game in the flappy_bird.py file, you will not get grade for it.**

**IMPORTANT: Most likely you will need to create additional classes. In this case, do NOT write classes in seperate files! Your submission should contain ONLY 1 or 2 .py files: 'snake.py' or/and 'flappy_bird.py'. If you decide to add custom sounds or images to your games, include all of them in the submission as well.**

*Note: Some images are attached to this assignment. They might give you a better idea, what you are asked to implement. But, this images are just examples, and you are not required to implement tasks in the exact same way.*

## Bonus

You can implement both of the games to obtain bonus points. As each game is worth of 10 points, you can get maximum 20 points (10 regular + 10 bonus) in this assignment. Based on your solution, it is possible to get partial points from both of them (For example, 7 points from Snake and 5 points from Flappy Bird). Bonus points can be used to compensate any lost points before the final exam.

## Plagiarism

**Plagiarism will not be tolerated!** Both of the games are part of the same assignment, therefore all 20 points will be lost if the assignment is nullified. The assignment can be nullified if you copy code from the internet or other students (Both, 1st year or 2nd year) and/or if the game does not follow rules described bellow.

# Snake

Snake is one of the oldest arcade games. Since 1976 dozens of versions have been implemented, but the main rules have not changed much.

*If you've never played it, you can google the 'snake', and the first thing you see will be Google's implementation of the game.*

## Game Rules

The game should follow the given rules:

- When the game starts snake should appear on the screen. At first, the snake should consist of 5 cells. (It's up to you how each cell is represented. For example, it can be 5 circles or 5 squares.)
- Snake should always be moving up, down, left, or right.
- At first snake should be moving to the right. But the user should be able to change the direction using the following keys:
  - `W` or `Up Arrow` - Snake goes up.
  - `S` or `Down Arrow` - Snake goes down.
  - `A` or `Left Arrow` - Snake goes left.
  - `D` or `Right Arrow` - Snake goes right.
- If the snake collides with the borders of the window game should end.

- If the snake collides with itself game should end.
- Food should appear on the board at random positions. You can represent food with a circle or with some pictures. Make sure that, food is not bigger than the head of the snake.
- There should be at most one food on the table at the same time.
- Snake eats food if its head collides with the food. Whenever a snake eats food its length should increase by one cell.

Those rules are mandatory. If your game does not follow them it may not be accepted. For example: if a snake can be moved with `AWSD` keys but not with arrows, or a completely different set of keys are used for movement, or a snake can go through walls and come on the opposite side. Even though this makes sense, it does not follow this assignment's rules and is not allowed.

This part of the game is essential, but it only gives you part of the points (See grading). The remaining points should be earned with extensions.

## Extentions

Below you can find some ideas and how many points they are worth. You can choose any subset of them and earned points will be equal to the sum of each extension's point.

**Important: In the beginning of the .py file add a single comment - a list of extentions implemented. You can copy extention describtion directly from the list given bellow.**

**Note: You can get at most 2 points with extensions, even if you do all of them you'll still get 2 points.**

**Note: Extensions are not counted if you get less than 3 points in the mandatory part.** For example You will not get points for fancy background and music if your snake does not move and food does not exist.

*Note: ordered extensions depend on each other. For example, You can not have score without a menu line and superfood without scoring. But, you can have a menu line without scoring and superfood.*

List of possible extensions and improvements:

- [0.5pt] Forbid 180 degree turns - If the snake is moving upwards and the player presses `S` or `Down Arrow` it should not be counted as collision and the snake should continue moving in the initial direction. These rules should apply to all 4 directions.
- Scoreboard
  1. [0.5pt] Menu line - Add a line in the upper part of the window. The line should be a new border. The game should end if the snake collides with it. Part of the window above the line can be used to display some data: game name, score, time...
  2. [1pt] Score - Add scoring system to the game. Every time snake eats food player earns some points which should be displayed at the upper part of the screen.
  3. [0.5pt] Super Food - Create a new type of food that appears rarely, gives the player a bit more points, and does not increase the snake's length.
- Ending Screen
  1. [1pt] End screen - Instead of closing the application on colliding, display the ending screen.
  2. [0.5pt] Replay - Ask if the player wants to replay the game and if so start over. For example: if the player presses the space bar, you can start the game again.
- Advanced Scoreboard - `Score` and `End Screen` are needed for this part.
  1. [0.5pt] Display Score - Display achieved a score on the end screen.
  2. [1.5pt] High Scores - Store top 10 high scores in a text file and display them on the end screen.
- Use images instead of figures
  - [0.5pt] Background - Use image for background.
  - [0.5pt] Food - Use image for food.
  - [0.5pt] Snake - Use image for snake.
- Audio
  - [0.5pt] Music - Add background music to the game.
  - [0.5pt] Crash sound - Add sound when collision happens.

    - [0.5pt] Mute - Add mute option. You can create a button or just use the `M` key. This one needs `Music` or `Crash sound`.
- Start Menu
    1. [1pt] Start - Instead of starting the game directly add a button/key for it. For example, Snake should start moving only after space is pressed. In such a case you should also display text to let the player know what to press to start the game.
    2. [1pt] Difficulties - Allow players to choose different difficulties. You can make the game more difficult by increasing speed or adding obstacles.

## Suggested implementation

**Note: This is only a suggestion. You can come up with a different approach as long as you use the given code template.**

Create a class for the snake. You can use a list to store snake cells. Each cell can be represented with a tuple of X and Y coordinate (It can either be the center of the circle or upper-left corner of the square or something else). In the constructor, add 5 cells to the list and try drawing the snake. To draw the snake you can traverse the list with the for loop and draw each cell separately. When the snake moves head moves by one cell and everything else follows. You can simulate this movement by adding a new cell at the beginning of the list and remove one from the end. To determine where the snake should move, you can add a direction attribute, which can have 4 different values: up, down, left, right. Based on the direction position of a new cell can be calculated. To control movement you only need to change the direction attribute. Check which key is pressed and change direction according to it. The next thing you can do is collision detection: you do not need to check the snake's whole body as only the head gets a new position it's enough to check whether the head collides with the border or any other part of the snake. You can create another class to describe food, use the simple circle to draw the food. To make sure that there's only one food on the map add a flag to the game, make it False if snake eats food and there's none left when the flag is False create new food, with random position and make the flag True.

## Grading

- Mandatory part
    - Rendering snake - 1pt
    - Moving snake - 2pt
    - Border collision - 1pt
    - Self collision - 1pt
    - Creating food - 1pt
    - Eating food - 2pt
- Extentions - 2pt

Partial points can be earned in the mandatory part. Points of extensions depend on the subset which you choose.

# Flappy Bird

Flappy bird was the most popular android and ios game released in 2013. Even though not much time has passed since releasing it (at least, compared to snake) there have been a lot of variations of the game and now you should add one more to the list.

*You can play games online version here - https://flappybird.io/*

## Game Rules

The game should follow the given rules:

- When the game starts bird should appear on the screen. At first, the bird should be positioned on the right side in the middle of the height. (For simplicity you can represent a bird with a circle or any other figure.)

- Bird should always be moving towards the right side of the screen with fixed speed.
- At first bird should be positioned in the middle of the height, but as soon as the game starts bird should start falling down. The bird should jump up when the user presses the `Space` key or clicks the `Mouse button`. *(Bounce application from the class should be a good example of jumping.)*
- There should be several pipes sticking from upper or lower borders. There should be at least 5 pipes. At least 2 pipes should be on each side. Pipes should not have the same length. (For simplicity you can represent pipes with lines or rectangles.)
- If the bird collides with the lower or upper borders of the window player loses and the game should end.
- If the bird collides with any of the pipes player loses and the game should end.
- If the bird collides with the right border of the window, the player has reached the end and wins the game.

Those rules are mandatory. If your game does not follow them it may not be accepted. For example: if the bird can be moved with `Space` but not with `Mouse Button`, or a completely different set of keys are used for movement. Even though this makes sense, it does not follow this assignment's rules and is not allowed.

This part of the game is essential, but it only gives you part of the points (See grading). The remaining points should be earned with extensions.

## Extentions

Below you can find some ideas and how many points they are worth. You can choose any subset of them and earned points will be equal to the sum of each extension's point.

**Important: In the beginning of the .py file add a single comment - a list of extentions implemented. You can copy extention describtion directly from the list given bellow.**

**Note: You can get at most 2 points with extensions, even if you do all of them you'll still get 2 points.**

**Note: Extensions are not counted if you get less than 3 points in the mandatory part.** For example You will not get points for fancy background and music if your bird does not move and does not collide with pipes.

*Note: ordered extensions depend on each other. For example you can not have coins without menu line.*

List of possible extensions and improvements:

- Better Visual
  - [0.5pt] Add background image
  - [0.5pt] Add bird image
  - [0.5pt] Add an image for pipes, do not forget to take care of different lengths.
- Menu
  1. [0.5pt] Menu line - Add a line in the upper part of the window. The line should be a new border. The game should end if the bird collides with it. Part of the window above the line can be used to display some data: game name, score, time...
  2. [1.0pt] Coins - Add coins on the random positions. Coins can be circles or images. If the bird collides with the coin then the user gets some point. Display points above the menu line.
- Ending
  1. [1pt] End screen - Instead of closing the application on colliding, display the ending screen with the result: win/lose.
  2. [0.5pt] Replay - Ask if the player wants to replay the game and if so start over. For example: if the player presses the space bar, you can start the game again.
- Audio
  - [0.5pt] Music - Add background music to the game.
  - [0.5pt] Crash sound - Add sound when collision happens.
  - [0.5pt] Mute - Add mute option. You can create a button or just use the `M` key. This one needs `Music` or `Crash sound`.
- Start Menu
  1. [1pt] Start - Instead of starting the game directly add a button/key for it. For example, the bird should start moving only after space is pressed. In such a case, you should also display text to let players know what to

press to start the game.
    2. [1pt] Difficulties - Allow players to choose different difficulties. You can make the game more difficult by increasing speed or adding obstacles.
- Infinite Game
    - [1.5pt] Instead of ending the game when a player crosses the right side, move the player on the left side (Y coordinate should stay the same), display a new set of pipes and continue the game.

## Suggested implementation

**Note: This is only a suggestion. You can come up with a different approach as long as you use the given code template.**

Create a separate class for the bird. Use the circle to draw the bird. At first add X speed to the bird, so that it moves towards the right border, next try to add gravity and jumping similar to what has been described during the class. After that, you can check border collisions, if the bird's coordinates go out of the window game should end. At last, add pipes to the game: a list can be used to store them and each pipe can be initialized during the init phase. Pipe's can have their class, or you can simply store their coordinates. During render traverse pipes list and draw each of them using line or rectangle or something else and during update check if the ball collides with any of them. Ball and pipe collision can be done with simple geometry.

## Grading

- Mandatory part
    - Rendering bird - 1pt
    - Rendering pipes - 1pt
    - Border collision - 1pt
    - Pipe collision - 2.5t
    - Jumping - 2.5pt
- Extentions - 2pt

Partial points can be earned in the mandatory part. Points of extensions depend on the subset which you choose.