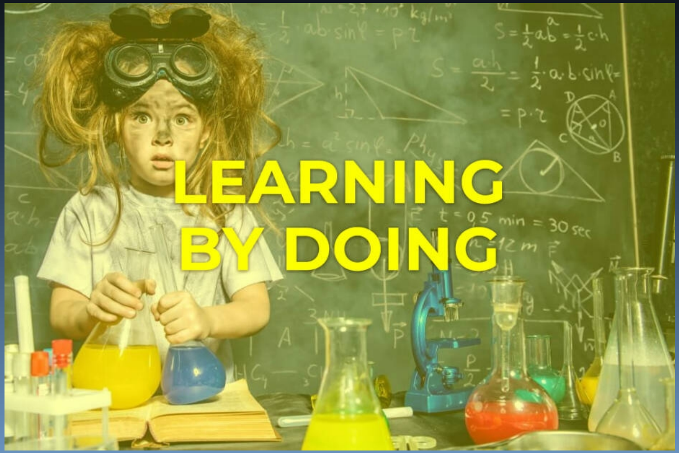
UNIVERSITÀ DI PARMA

Snake Game



For the things we have to learn before we can do them, we learn by doing them

Aristotle, The Nicomachean Ethics

You don't learn to walk by following rules. You learn by doing, and by falling over

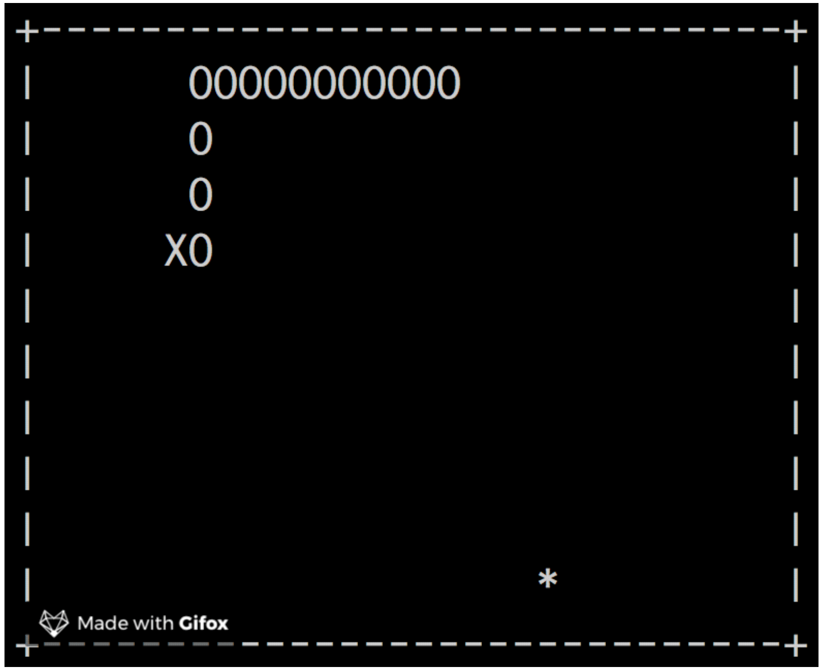
Richard Branson, Virgin Group

Summary

- Why this game?
- Prerequisites
- Different development steps
  - Flow control
  - Variables and constants
  - Predefined functions
  - Arrays
  - Functions
  - I/O

Why this game?

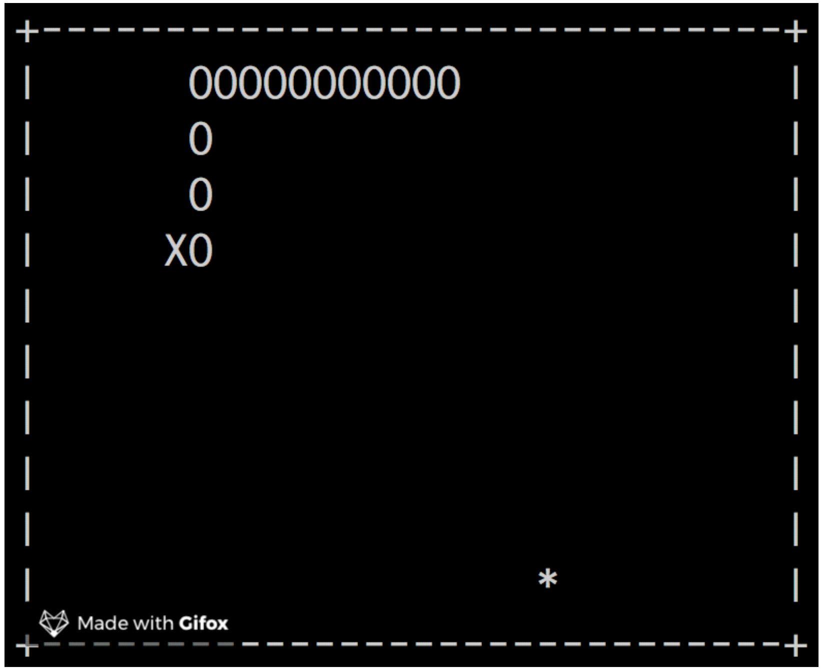
- The idea is giving you a project that will be continuously improved up to the end of this course
- Console based graphic
  - Back to '70s
- Please note that these slides will be continuously updated!



```
+-----------------------------+
|       00000000000           |
|       0                     |
|       0                     |
|      X0                     |
|                             |
|                             |
|                             |
|                             |
|                             |
|                     *       |
+-----------------------------+
```

Why this game?

- What is the snake progam?
- It is a simple game that can easily be rendered using only the console (1976)
- The player control a "snake" that is continuously moving
- Each time the "head" of the snake "eat" some "food", the snake itself grows becoming longer and longer
- When the snake hits the "walls" or itself it is a game over!



```
+-------------------------+
|      00000000000        |
|      0                  |
|      0                  |
|     X0                  |
|                         |
|                         |
|                         |
|                         |
|                         |
|                 *       |
+-------------------------+
```

Prerequisites

- Even console “graphics” can be difficult
- 2 main issues:
  - In a game I would like to read keypresses without stopping the program
    - scanf() and other console input functions stops until a “return” is entered…
  - I also would like to be able to write in a given position
    - printf() and other console output functions can not behave like this…

Prerequisites

- We will start from a C “skeleton” that contains specific functions to solve those issues
- Therefore:
  - Create a “snake” project in your folder (U:\ for Lab workstations)
  - Overwrite the main.c in your project with the skel.c in the “snake” folder in the lab repository
  - Try to compile & run

10. Flow control: selection

- Print a menu like:
  1. Start game
  2. Scores
  3. Help
  4. Quit
- Read from keyboard the user choice

10b. Flow control: selection

- Print the choice (as debug) and act as follows
  - Help:
    - Print “Q: left, E: right, W: up, S: down, ‘space’: pause, X: quit”
  - Quit: end the program
  - Scores: do nothing for now
  - Start: go on to the next slide
- What is the best selection statement to use?

20. Flow Control: cycles

- Modify step #10 in order to print again the menu:
  - When the user enters a wrong choice
  - After the help
  - Later, also after the start…
- When the user enters the “start” go on according to next slides

20b. Flow Control: cycles

- Print the game field
- Namely, a **23×97** rectangle surrounded by a border
  - Use '+' for corners
  - Use '-' for horizontal borders
  - Use '|' (ASCII 124) for vertical borders
- Use a top down approach!

20b. Flow Control: cycles

- Hints for printing the playing field
  - Do not stick on a given size for the field
    - Put the 23 and 97 in variables → "generalize" your approach
  - Use a top down approach
    - Imagine yourself driving down the field using the limitations of the printf()
    - What we have to write initially? And then? How many times?
- Questions:
  - How many cycles do we need?
  - Do we need to nest some cycles?

30. Variables & Constant

- Use a variable for the score and init it to zero
  - What is the best type to use?
- Modify 20#:
  - Print the score before the playing field using 6 characters and using zero as padding character
  - Use constants for the symbols used for the border of the playing field
  - Use constants for the size of the playing field
- Define also variables for the snake head position and heading
  - Initialize them with random values
  - Which kind of variables do we have to use? How many?

40. Predefined functions: usleep()

- Study the following functions (on the book!)

  #include <unistd.h>

  int usleep(usec);

- It stops the program execution for usec µs (1 µs = 1000 ms)
- It can be used during the game (later)

40. Predefined functions

- In the skeleton you can find other functions
- They are **NOT** C predefined functions
- But they have been already fully defined
- Then we can use them as predefined functions anyway

40. “Predefined” functions: clearscreen()

UNIVERSITÀ DI PARMA

```
void clearscreen(void)
```

- It “clears” the console
- Then it can be used when starting the program, when switching back to the menu after playing, …

40. “Predefined” functions: gotoxy()

void gotoxy(int, int)

- When using printf() and similar functions, the output is written in a constrained position in the console
  - “printing cursor” position
  - From left to right, from the top to bottom
- In a game I have to be more free about printing position
- The gotoxy() function allow to move the cursor in a given position
  - row and column coordinates
  - The top-left position in the console has (∅, ∅) coordinates
  - Column index increases moving right
  - Row index increases moving down

40. “Predefined” functions: getcommand()

char getcommand(void)

- scanf() and similar keyboard input functions “stop” the execution
  - They wait for a “enter/return”
- In a game we need to be able to read keystrokes in a non blocking fashion
- getcommand() returns the ASCII value of a key that has been pushed or 0 if no key has been stroke

50. Start the game!

- In #30 you defined and randomly initialized the variables for storing the snake position and movement
- Exploiting the gotoxy() functions:
  - Write the snake head 'X' in the console!

60. Update the game!

- After printing the playing field & the snake head:
  - Use an infinite loop for continuously updating the console
  - Each cycle:
    - Update the snake head position according to the randomly chosen direction where the snake is heading
    - Check whether the snake hits a border, in such a case exit the infinite loop
- Which instructions I can use for an “infinite” loop?
- How can I exit such a loop?

60. Update the game! (2)

- In #30 you should have defined which kind of variables to use for storing snake head position and its heading
  - For position it is fine to use 2 variables for the 2 coordinates
  - But what is the best way to store the direction where the snake is heading?
- Most of you use a single variable to encode the direction
  - e.g. 1 for heading left, 2 for right etc.

60. Update the game! (2)

- This choice can be effective but it is not so efficient
  - Each time we have to "transcode" the value
- Alternative solution:
  - The heading can be seen as a vector
  - Encode horizontal & vertical components
    - Pros:
      - Movements $\longrightarrow$ directly encoded
      - Change of direction $\longrightarrow$ easy
    - Cons:
      - We need 2 variables

60. Update the game! (3)

- A simple 90° rotation, can be used to update the ſtate when a "turn" is requeſted

$$\begin{pmatrix} nm_x \\ nm_y \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix} \qquad \begin{pmatrix} nm_x \\ nm_y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix}$$

70. read the input

- Use the char getcommand(void) function to read the keyboard input
- Move the snake head accordingly:
  - Q: left
  - E: right
  - W: up
  - S: down

80. add some food & poison

- Randomly (i.e. not each cycle) add some food ('#') and some poison ('Y') in a random position (i.e. random column/row) in the playing field
- Increase/Decrease the player score when the snake head get it!
- Just one piece of food and one piece of poison (until the snake eats it!)

90. arrays: add snake body

- When the snake head hits the food:
  - Food must be removed from the playing field
  - A snake body grows… "OOOOO"
    - Initially no body
    - Each time he gets food add a piece of body 'O'
- For poison implement an opposite behavior
- Use arrays for dealing with the body
  - A single array or multiple ones and what size?
  - Which kind of data I need?
  - Arrays are enough?

100. Dynamic memory allocation

- Use a dynamically allocated array to deal with snake body
  - Use malloc() to allocate first body piece
  - Use realloc() to grow/reduce it piece by piece
  - Use free() when the game ends

120. User-defined functions

- Factorize your code using functions, e.g.:
  - menu()
  - printfield()
  - printsnake()
  - putfood()
  - putpoison()
  - updatefield()
  - …
- Barely nothing other than function invocations in the main()!

130. files

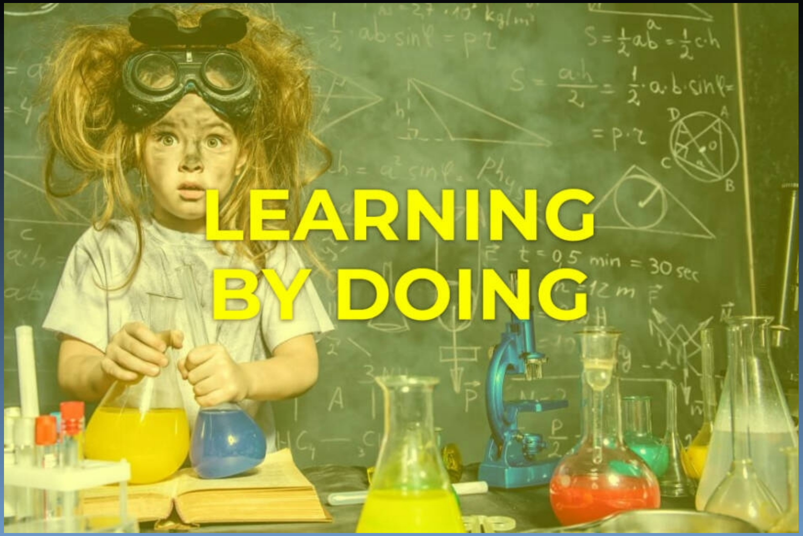
- When a game session ends, ask the user about his nickname and save the score in a CSV files, like
  - Nickname;date;score
  - Use an “append” strategy for saving data
- When user select “S” from Menu
  - Print:
    - Last score
    - Best score

140. ſtruct

- Add ſtruct to your code
  - i.e. for managing snake data
  - Does this simplify function calls?

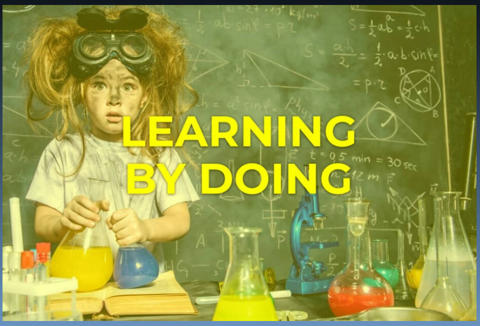
UNIVERSITÀ DI PARMA

The End (so far...)



UNIVERSITÀ DI PARMA

Snake Game



For the things we have to learn before we can do them, we learn by doing them

Aristotle, The Nicomachean Ethics

You don't learn to walk by following rules. You learn by doing, and by falling over

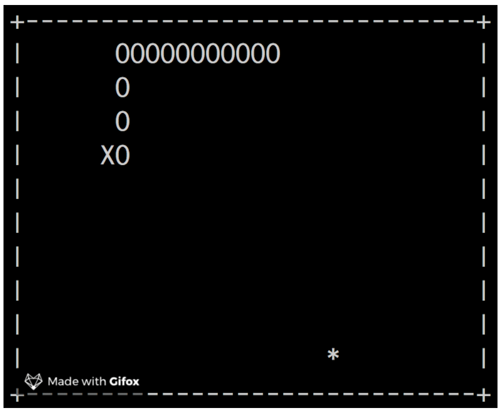
Richard Branson, Virgin Group

Summary

- Why this game?
- Prerequisites
- Different development steps
  - Flow control
  - Variables and constants
  - Predefined functions
  - Arrays
  - Functions
  - I/O

Why this game?

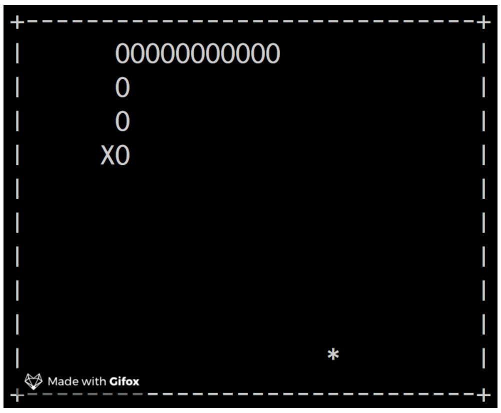
- The idea is giving you a project that will be continuously improved up to the end of this course
- Console based graphic
  - Back to '70s
- Please note that these slides will be continuously updated!



```
+-----------------------------+
|      00000000000            |
|      0                      |
|      0                      |
|     X0                      |
|                             |
|                             |
|                             |
|                             |
|                             |
|                   *         |
+-----------------------------+
```

Why this game?

- What is the snake progam?
- It is a simple game that can easily be rendered using only the console (1976)
- The player control a "snake" that is continuously moving
- Each time the "head" of the snake "eat" some "food", the snake itself grows becoming longer and longer
- When the snake hits the "walls" or itself it is a game over!



```
+----------------------------+
|      00000000000           |
|      0                     |
|      0                     |
|     X0                     |
|                            |
|                            |
|                            |
|                            |
|                            |
|                  *         |
+----------------------------+
```

Prerequisites

- Even console “graphics” can be difficult
- 2 main issues:
  - In a game I would like to read keypresses without stopping the program
    - scanf() and other console input functions stops until a “return” is entered…
  - I also would like to be able to write in a given position
    - printf() and other console output functions can not behave like this…

Prerequisites

- We will start from a C “skeleton” that contains specific functions to solve those issues
- Therefore:
  - Create a “snake” project in your folder (U:\ for Lab workstations)
  - Overwrite the main.c in your project with the skel.c in the “snake” folder in the lab repository
  - Try to compile & run

10. Flow control: selection

UNIVERSITÀ DI PARMA

- Print a menu like:
    1. Start game
    2. Scores
    3. Help
    4. Quit
- Read from keyboard the user choice

10b. Flow control: selection

UNIVERSITÀ DI PARMA

- Print the choice (as debug) and act as follows
  - Help:
    - Print “Q: left, E: right, W: up, S: down, ‘space’: pause, X: quit”
  - Quit: end the program
  - Scores: do nothing for now
  - Start: go on to the next slide
- What is the best selection statement to use?

20. Flow Control: cycles

- Modify step #10 in order to print again the menu:
  - When the user enters a wrong choice
  - After the help
  - Later, also after the start…
- When the user enters the “start” go on according to next slides

20b. Flow Control: cycles

- Print the game field
- Namely, a **23×97** rectangle surrounded by a border
  - Use '+' for corners
  - Use '-' for horizontal borders
  - Use '|' (ASCII 124) for vertical borders
- Use a top down approach!

20b. Flow Control: cycles

UNIVERSITÀ DI PARMA

- Hints for printing the playing field
  - Do not stick on a given size for the field
    - Put the 23 and 97 in variables → “generalize” your approach
  - Use a top down approach
    - Imagine yourself driving down the field using the limitations of the printf()
    - What we have to write initially? And then? How many times?
- Questions:
  - How many cycles do we need?
  - Do we need to nest some cycles?

30. Variables & Constant

UNIVERSITÀ DI PARMA

- Use a variable for the score and init it to zero
  - What is the best type to use?
- Modify 20#:
  - Print the score before the playing field using 6 characters and using zero as padding character
  - Use constants for the symbols used for the border of the playing field
  - Use constants for the size of the playing field
- Define also variables for the snake head position and heading
  - Initialize them with random values
  - Which kind of variables do we have to use? How many?

40. Predefined functions: usleep()

- Study the following functions (on the book!)

  #include <unistd.h>

  int usleep(usec);

- It stops the program execution for usec μs (1 μs = 1000 ms)
- It can be used during the game (later)

40. Predefined functions

- In the skeleton you can find other functions
- They are **NOT** C predefined functions
- But they have been already fully defined
- Then we can use them as predefined functions anyway

40. “Predefined” functions: clearscreen()

UNIVERSITÀ DI PARMA

void clearscreen(void)

- It “clears” the console
- Then it can be used when starting the program, when switching back to the menu after playing, ...

40. “Predefined” functions: gotoxy()

void gotoxy(int, int)

- When using printf() and similar functions, the output is written in a constrained position in the console
  - “printing cursor” position
  - From left to right, from the top to bottom
- In a game I have to be more free about printing position
- The gotoxy() function allow to move the cursor in a given position
  - row and column coordinates
  - The top-left position in the console has (0, 0) coordinates
  - Column index increases moving right
  - Row index increases moving down

40. “Predefined” functions: getcommand()

char getcommand(void)

- scanf() and similar keyboard input functions “stop” the execution
  - They wait for a “enter/return”
- In a game we need to be able to read keystrokes in a non blocking fashion
- getcommand() returns the ASCII value of a key that has been pushed or Ø if no key has been stroke

50. Start the game!

- In #30 you defined and randomly initialized the variables for storing the snake position and movement
- Exploiting the gotoxy() functions:
  - Write the snake head ‘X’ in the console!

60. Update the game!

- After printing the playing field & the snake head:
  - Use an infinite loop for continuously updating the console
  - Each cycle:
    - Update the snake head position according to the randomly chosen direction where the snake is heading
    - Check whether the snake hits a border, in such a case exit the infinite loop
- Which instructions I can use for an “infinite” loop?
- How can I exit such a loop?

60. Update the game! (2)

- In #30 you should have defined which kind of variables to use for storing snake head position and its heading
  - For position it is fine to use 2 variables for the 2 coordinates
  - But what is the best way to store the direction where the snake is heading?
- Most of you use a single variable to encode the direction
  - e.g. 1 for heading left, 2 for right etc.

60. Update the game! (2)

- This choice can be effective but it is not so efficient
  - Each time we have to “transcode” the value
- Alternative solution:
  - The heading can be seen as a vector
  - Encode horizontal & vertical components
    - Pros:
      - Movements $\longrightarrow$ directly encoded
      - Change of direction $\longrightarrow$ easy
    - Cons:
      - We need 2 variables

60. Update the game! (3)

- A simple 90º rotation, can be used to update the ſtate when a “turn” is requeſted

$$\begin{pmatrix} nm_x \\ nm_y \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix} \qquad \begin{pmatrix} nm_x \\ nm_y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix}$$

70. read the input

- Use the char getcommand(void) function to read the keyboard input
- Move the snake head accordingly:
  - Q: left
  - E: right
  - W: up
  - S: down

80. add some food & poison

UNIVERSITÀ DI PARMA

- Randomly (i.e. not each cycle) add some food ('#') and some poison ('Y') in a random position (i.e. random column/row) in the playing field
- Increase/Decrease the player score when the snake head get it!
- Just one piece of food and one piece of poison (until the snake eats it!)

90. arrays: add snake body

- When the snake head hits the food:
  - Food must be removed from the playing field
  - A snake body grows… “OOOOO”
    - Initially no body
    - Each time he gets food add a piece of body ‘O’
- For poison implement an opposite behavior
- Use arrays for dealing with the body
  - A single array or multiple ones and what size?
  - Which kind of data I need?
  - Arrays are enough?

100. Dynamic memory allocation

UNIVERSITÀ DI PARMA

- Use a dynamically allocated array to deal with snake body
  - Use malloc() to allocate first body piece
  - Use realloc() to grow/reduce it piece by piece
  - Use free() when the game ends

120. User-defined functions

- Factorize your code using functions, e.g.:
  - menu()
  - printfield()
  - printsnake()
  - putfood()
  - putpoison()
  - updatefield()
  - …
- Barely nothing other than function invocations in the main()!

130. files

- When a game session ends, ask the user about his nickname and save the score in a CSV files, like
  - Nickname;date;score
  - Use an "append" strategy for saving data
- When user select "S" from Menu
  - Print:
    - Last score
    - Best score

140. ſtruct

UNIVERSITÀ DI PARMA

- Add ſtruct to your code
  - i.e. for managing snake data
  - Does this simplify function calls?

UNIVERSITÀ DI PARMA

The End (so far...)