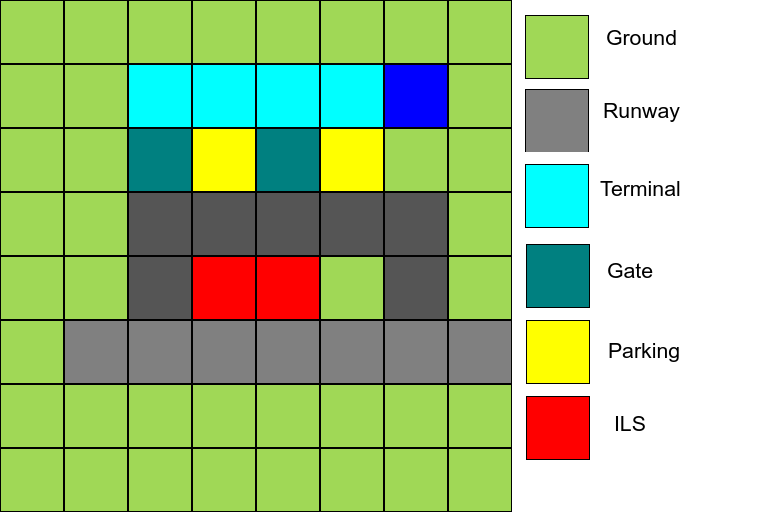
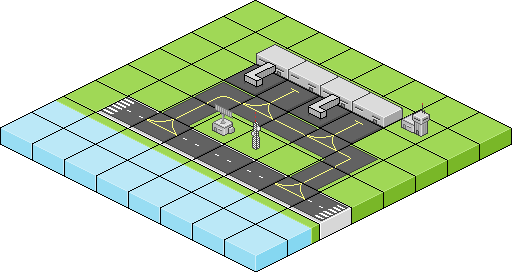
# Level design



First levels consist of a 16x16 square grid (represented as 8x8 grid below for convenience). More difficult levels would feature 16x16 or 32x32 square grids. It should take between 256 bytes and 1024 bytes.



Levels are to be stored in a custom file format called \*.LVL.

# Level format \*.LVL

* Header
  + 0x0 to 0x1F: Airport (or level) name string. Example: “Test Airport”. Max 32 characters, null character included.
  + 0x20: Map width in tiles. For example: 16d
  + 0x21: Map height in tiles. For example: 16d
  + 0x22 – 0x29: Runway edges (max. 4 runways)
    - 0x22 – 0x23: Runway 1
    - 0x24 – 0x25: Runway 2
    - 0x26 – 0x27: Runway 3
    - 0x28 – 0x29: Runway 4
  + 0x2A – 0x2E: Reserved (zero filled).
  + 0x2F: Map file checksum (all previous bytes *XORed*)
* Data buffer
  + 0x30 – 0x42F: Level data
* Aircraft data:

***32 aircraft entities max. Memory start address: 0x430. Size for each entry: 16 bits.***

* + For each aircraft entry (relative position):
    - 0x0 – 0x7: Flight number string. Example: “UN8631”. Max 8 characters, null character included.
    - 0x8: Airline ID.
    - 0x9 – 0xA: Seconds to spawn. Example: 2 min + 35 sec -> 155d.
    - 0xB: Passengers.
    - 0xC – 0xF: Reserved. Zero filled.

# Data types definitions

* Byte -> 8 bits. Range from -128 to 127.
* Half -> 16 bits. Range from -32768 to 32767.
* Word -> 32 bits. Range from -2147483648 to 2147483647.
* Fix16\_t -> 32 bits (16.16 fixed-point format)

# Aircraft structure

* Airline ID. Unsigned Byte (see AIRCRAFT\_AIRLINE\_ID\_ENUM section).
* Tile. Signed half.
* X pos (relative to tile). Signed Half
* Y pos. Signed Half
* Z pos. Signed Half
* Direction. Unsigned Byte. See (*AIRCRAFT\_DIRECTION\_ENUM* section).
* State. Unsigned Byte (See *AIRCRAFT\_STATE\_ENUM* section).
* Approach timer. Unsigned half.
* Speed. Unsigned byte.
* Passengers. Unsigned byte.
* Combination [4]. Unsigned half.

# AIRCRAFT\_STATE\_ENUM

* AIRCRAFT\_STATE\_NULL = 0
* AIRCRAFT\_STATE\_APPROACH
* AIRCRAFT\_STATE\_PARKED
* AIRCRAFT\_STATE\_LANDING
* AIRCRAFT\_STATE\_LANDED
* AIRCRAFT\_STATE\_TAXI
* AIRCRAFT\_STATE\_FINISHED

# AIRCRAFT\_DIRECTION\_ENUM

* AIRCRAFT\_DIRECTION\_UP = 0
* AIRCRAFT\_DIRECTION\_RIGHT
* AIRCRAFT\_DIRECTION\_DOWN
* AIRCRAFT\_DIRECTION\_LEFT

# AIRCRAFT\_AIRLINE\_ID\_SECTION

* AIRLINE\_NONE = 0
* AIRLINE\_ONE

# Gamepad key list

* PlayStation
  + KEY\_FLIGHT\_LIST: PAD\_CIRCLE
  + KEY\_CONFIRM: PAD\_CROSS
  + KEY\_CANCEL: PAD\_TRIANGLE
  + KEY\_UP: PAD\_UP
  + KEY\_DOWN: PAD\_DOWN
  + KEY\_LEFT: PAD\_LEFT
  + KEY\_LEFT: PAD\_RIGHT

# Core gameplay

Aircraft (from small aircraft to big airliners) with different number of passengers shall arrive to the airport and request the air traffic controller (from now on, ATC or player) to land on it. The main objective is to attend final approach, landing, taxiing and parking procedures.

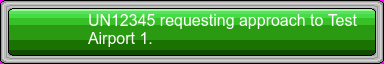
An aircraft arrives the ATC range when:

* Aircraft.Approach\_Timer expires

Also, this should change the following state:

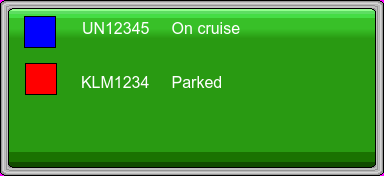
* Aircraft.State = AIRCRAFT\_STATE\_APPROACH

When an aircraft arrives to a certain miles far from the airport, an approach request is submitted:

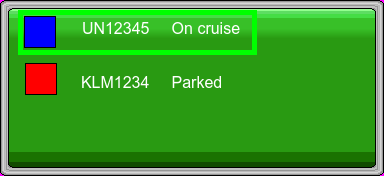


*An airport chatter sound may play.*

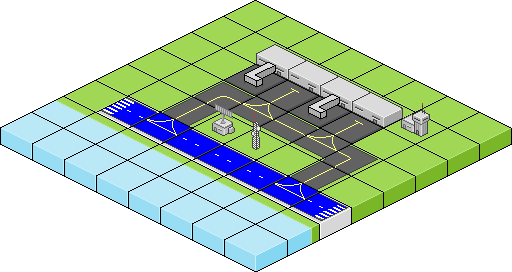
Flights for each level are listed on the LVL file. Player can read this list pressing KEY\_FLIGHT\_LIST and a window will appear. Current state for all aircraft is then shown:



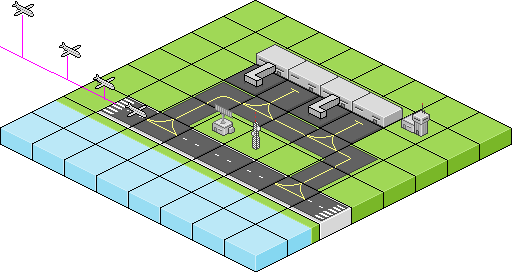
This list can be used to select next action for aircraft requests. For example, imagine UN12345 asks for an approach. The player then must assign a runway for UN12345. Now, the player has to click KEY\_FLIGHT\_LIST and select flight number UN12345.



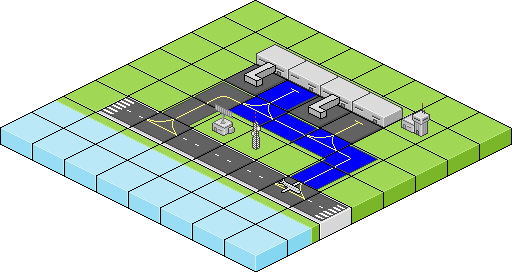
Once selected, possible runways shall be highlighted in blue (runway turns blue and black continuously).



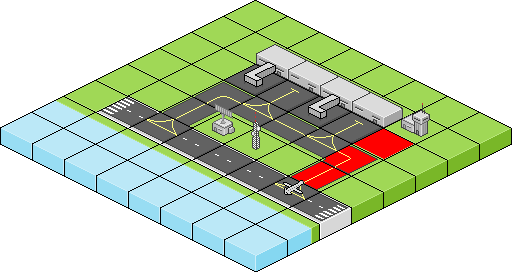
Once the runway has been selected by the player, the aircraft will appear a few tiles away from it, descending continuously.



Once landed, the aircraft will continue running through the runway until it reaches an intersection. At this point, a taxi request is made by the aircraft and waits until the player tells what taxi points it should follow).

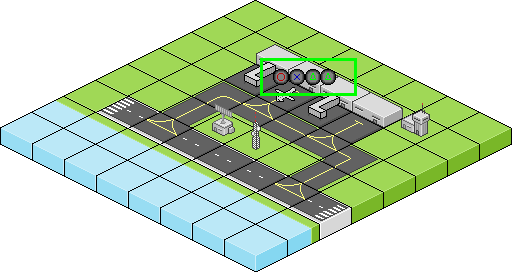


Waypoints are selected through multiple straight lines. If a wrong tile is selected (i.e.: grass), the whole track becomes red.



The plane will park on the selected parking. Then, passengers have to be unloaded. A 4-key combination (in this example, 4 PlayStation buttons) has to be pressed for a certain number of passengers. For bigger airplanes, combinations may have to be solved repeatedly. For example, a 120-passenger aircraft may require solving up to 4 different combinations.

Each time a combination is resolved, player will earn some points.



# How to win

The player needs to accomplish a certain score to win. The player earns score when unloading passengers from an aircraft, once parked. Delaying a flight makes the player to lose money. All aircraft must be parked to finish the level.

# Software design

## Abstraction layers description

## Main()

## Init()

Main() section is mainly divided into two sections: Init and Main loop, which is entered every 20 ms for PAL games (or 16,67 ms for NTSC games).

Rough example:

void GameHandler**(**void**)**

**{**

/\* Before, calls to another Sep layer routines \*/

**if(**Ev**.**MovePlayerUp**)**

**{**

SepMovePlayerUp**();**

**}**

**}**

void SepMovePlayerUp**(**void**)**

**{**

/\* Here, instructions regarding moving player up \*/

#if \_\_PSXSDK\_\_

GsSortSprite**(&**SDK**.**Sprites**.**Player**);**

#endif

#if \_\_SDL\_\_

/\* SDL equivalent for GsSortSprite() \*/

#endif

**}**

# Game Functions

## GameCartesianToIsometricPos()

#define TILE\_WIDTH 64

#define TILE\_WIDTH\_HALF 32

#define TILE\_HEIGHT 64

#define TILE\_HEIGHT\_HALF 32

struct t\_CartToIso

{

u8 u8TileX;

u8 u8TileY;

s16 s16IsoX;

s16 s16IsoY;

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* void GameCartesianToIsometricPosition(TYPE\_CartToIso \* CTI)

\*

\* BRIEF:

\* \* Transforms (X,Y,Z) position buffer to Isometric position

\*

\* PARAMETERS:

\* \* TYPE\_GXYZTOISO \* t\_GXYZTOISO

\*

\* RETURNS:

\*

\* \* void

\*

\* DATE:

\* 1/January/2016

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\*

\* screen.x = map.x \* TILE\_WIDTH\_HALF - map.y \* TILE\_WIDTH\_HALF;

screen.y = map.x \* TILE\_HEIGHT\_HALF + map.y \* TILE\_HEIGHT\_HALF;

\* \*/

void GameCartesianToIsometricPosition(TYPE\_CartToIso \* CTI)

{

CTI->u16IsoX = (CTI->u8TileX \* TILE\_WIDTH\_HALF) - (CTI->u8TileY \* TILE\_WIDTH\_HALF);

CTI->u16IsoY = (CTI->u8TileX \* TILE\_HEIGHT\_HALF) - (CTI->u8TileY \* TILE\_HEIGHT\_HALF)

}