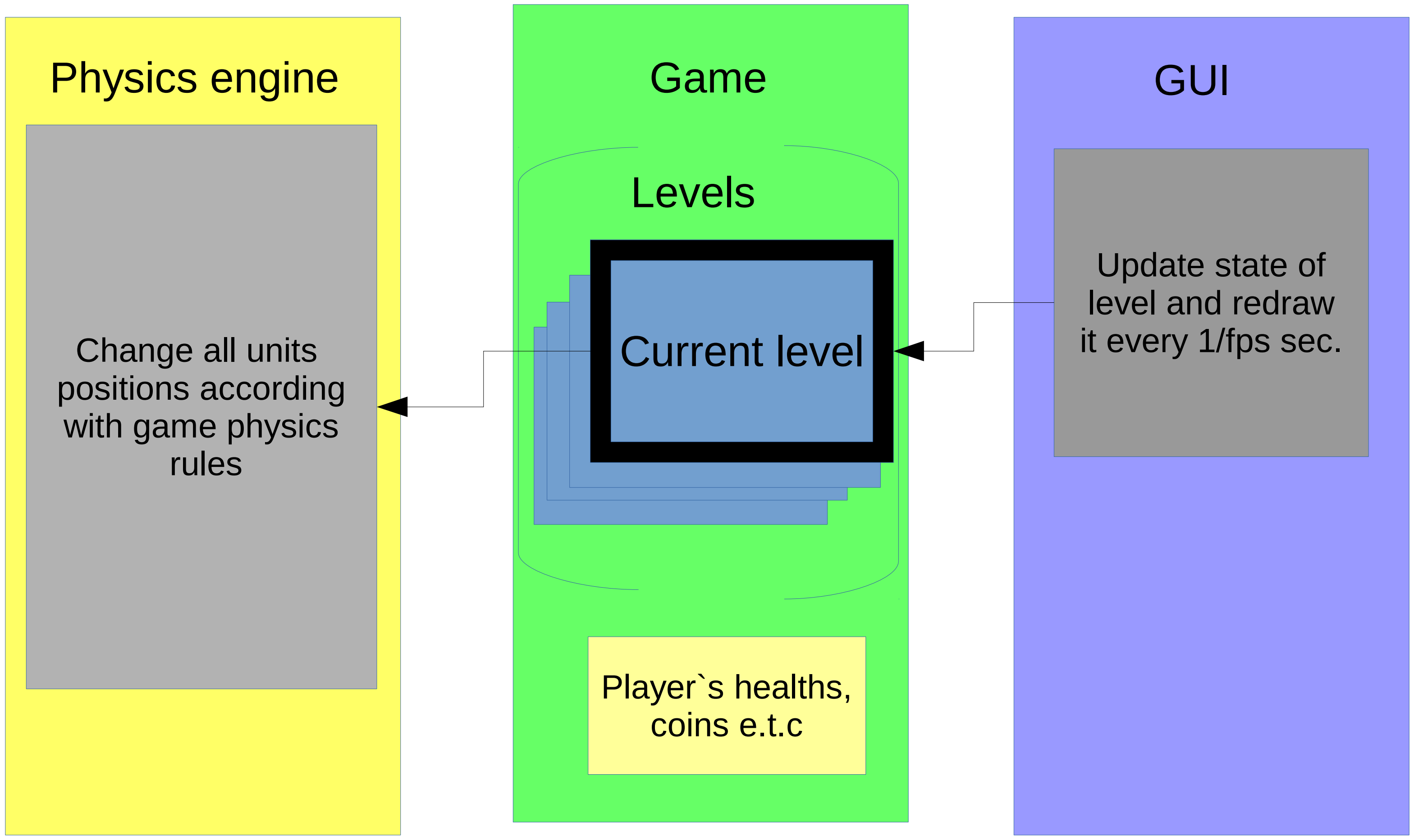
Our game project

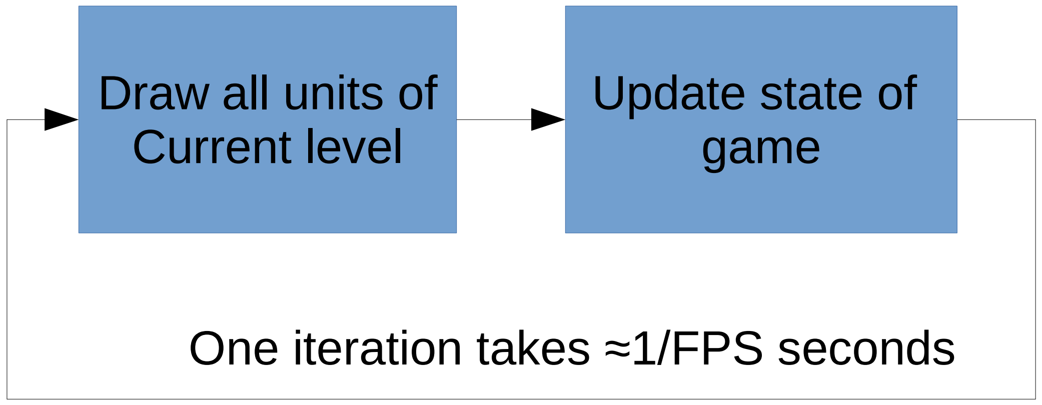
- Modules of the project
- Interaction with player
- Unit structure
- Collision matching
- Collision events

Modules of the project


Physics engine
Change all units positions according with game physics rules
Game
Levels
Current level
Player`s healths, coins e.t.c
GUI
Update state of level and redraw it every 1/fps sec.

Modules of the project - GUI

- GUI layer of the project does not know about any details of game. It has only API of game which can give for GUI coordinates of rectangles of all units to drawing and textures associated with every drown rectangle.
- Also GameAPI has function to update states of units by one frame.
- GUI draws all rectangles in the level, fill it by associated textures and call function to update state of the game. After it GUI repeats this sequence of actions by every 1/FPS seconds.

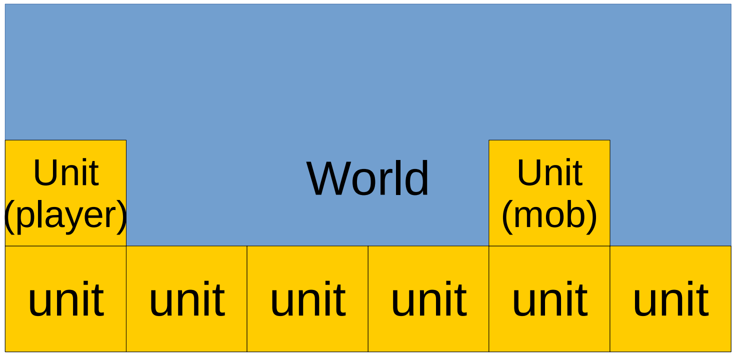

Draw all units of Current level
Update state of game
One iteration takes ≈1/FPS seconds

Modules of the project - Game

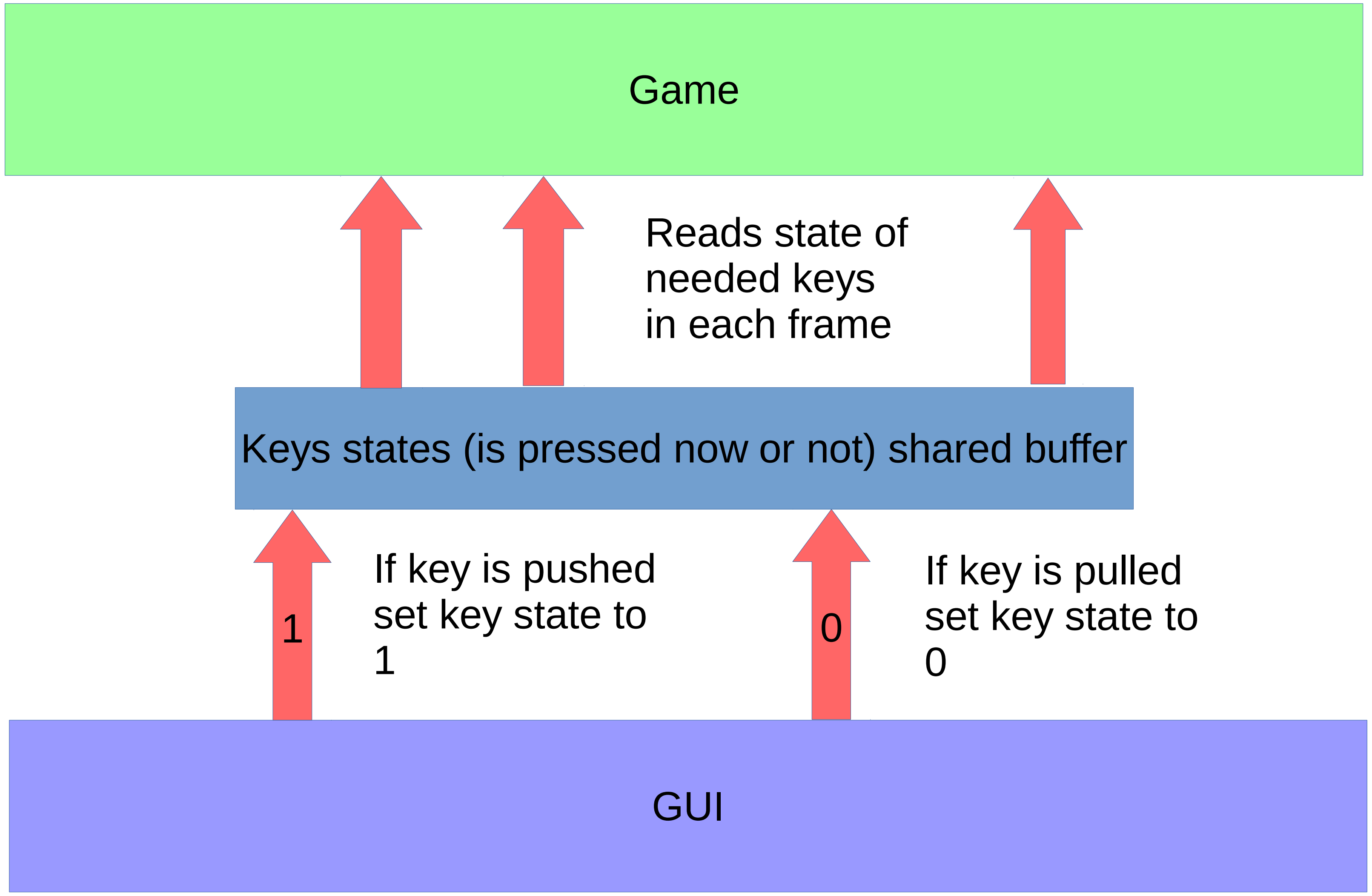
- Game layer contains levels, player`s statistics and data like a coins, health, e.t.c.
- Game has API for GUI: in our implementation of this architecture game implements GameAPI class.
- In our implementation of the Game it is class Game and interface GameAPI which is implemented by this class. Also in our implementation Game`s constructor takes reference for shared buffer of keys states (see Interaction with player).

Modules of the project — Physics engine

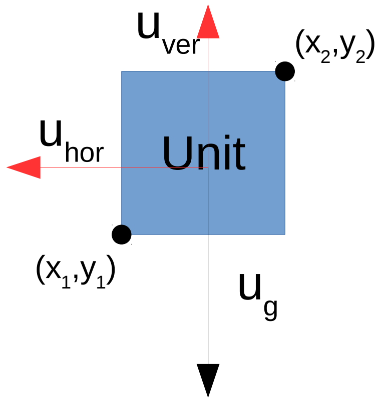
- Levels are objects of World class which represents one world`s items and functions which describe relations between them.
- World have function for updating states of all it`s items according to physics rules which are described by internal world functions.
- World consists of units: every game item is unit in the world.



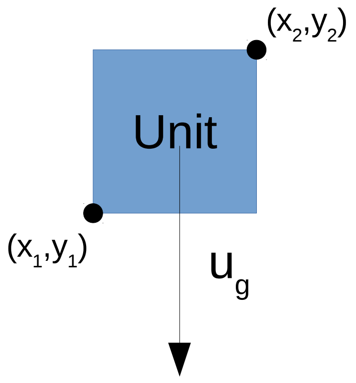
Interaction with player


Game
Reads state of needed keys in each frame
Keys states (is pressed now or not) shared buffer
1
If key is pushed set key state to 1
0
If key is pulled set key state to 0
GUI

Unit structure



$$[\vec{u}] = \frac{\vec{points}}{frame}$$



Size and position of unit can be described by two points: left bottom (x_1,y_1) ant top right (x_2,y_2).

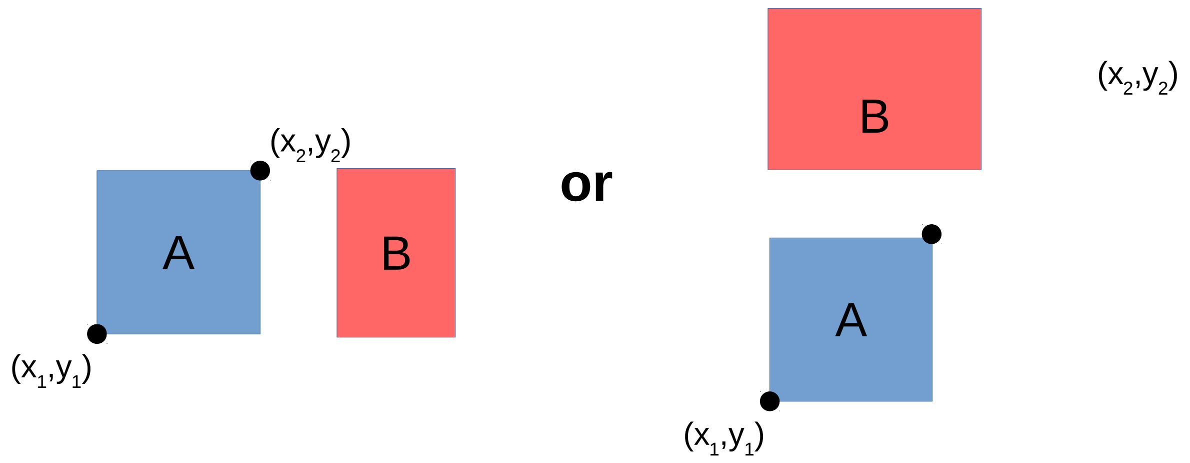
For describe current state of the unit in the world it’s enough to have a two current speed vectors: vertical and horizontal. Measure of speed in our game context is points per frame. Each frame = n seconds. Also there is speed given by gravitation, and it always force unit.

If unit is alone with no other units, it has only vertical gravitation speed, witch make unit move down.

If two blocks bumped, block with lower priority takes speed of block with higher priority. So each unit should have priority. Units of ground and walls normally have the largest priority.

Collision matching

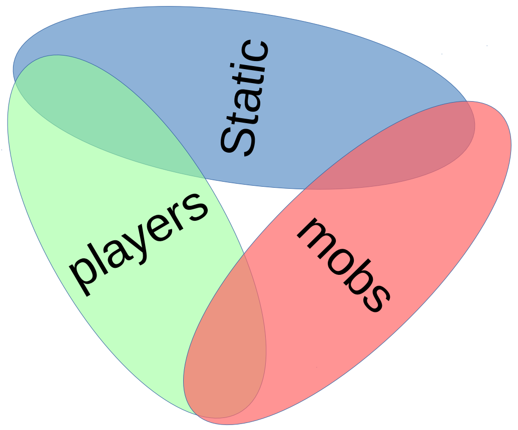
Position of A < position of B:

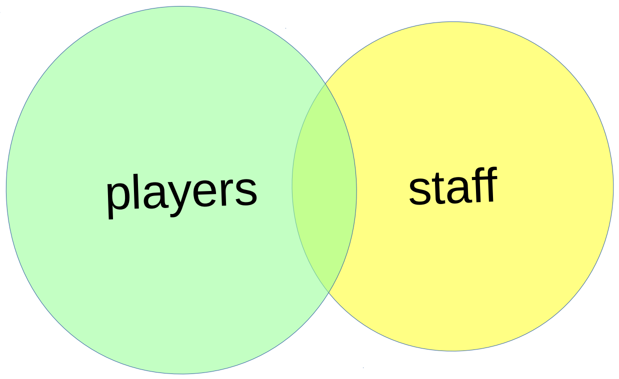


Unit A position is less then unit B position if end of A by x is less then
begin of B by x or top side of A is less then bottom
side of B. In the other cases collision has place.
In the engine it means that if **NOT(A < B) AND NOT(B < A)**
then they are collised.

Collision matching

- There are three sets of objects: static, mobs and players.
- Collisions are matched only between different sets. It greatly decrease number of calculations of interactions between units.
- There is additional 4th group named staff. It contains units which should interact only with players, e.g. coins.

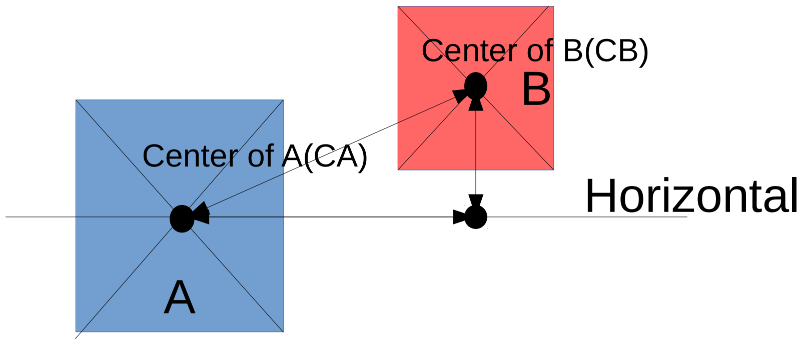


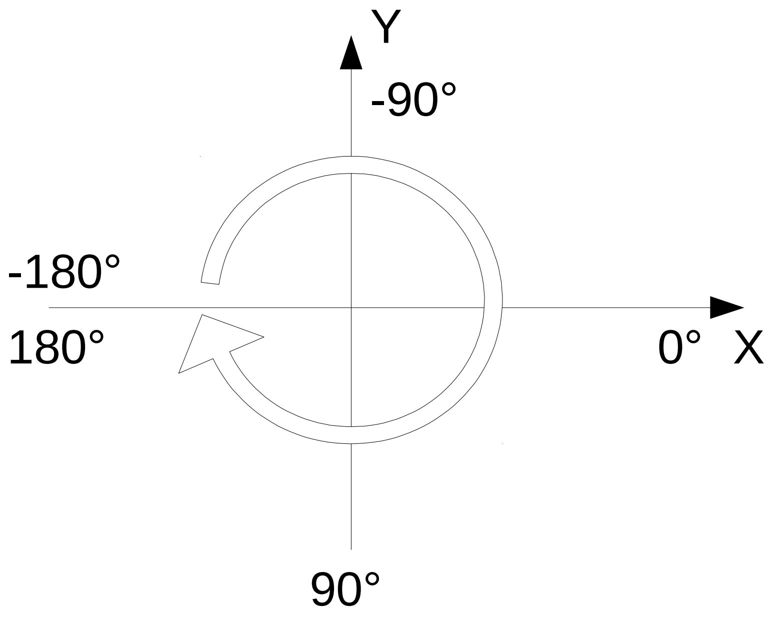


Collision matching

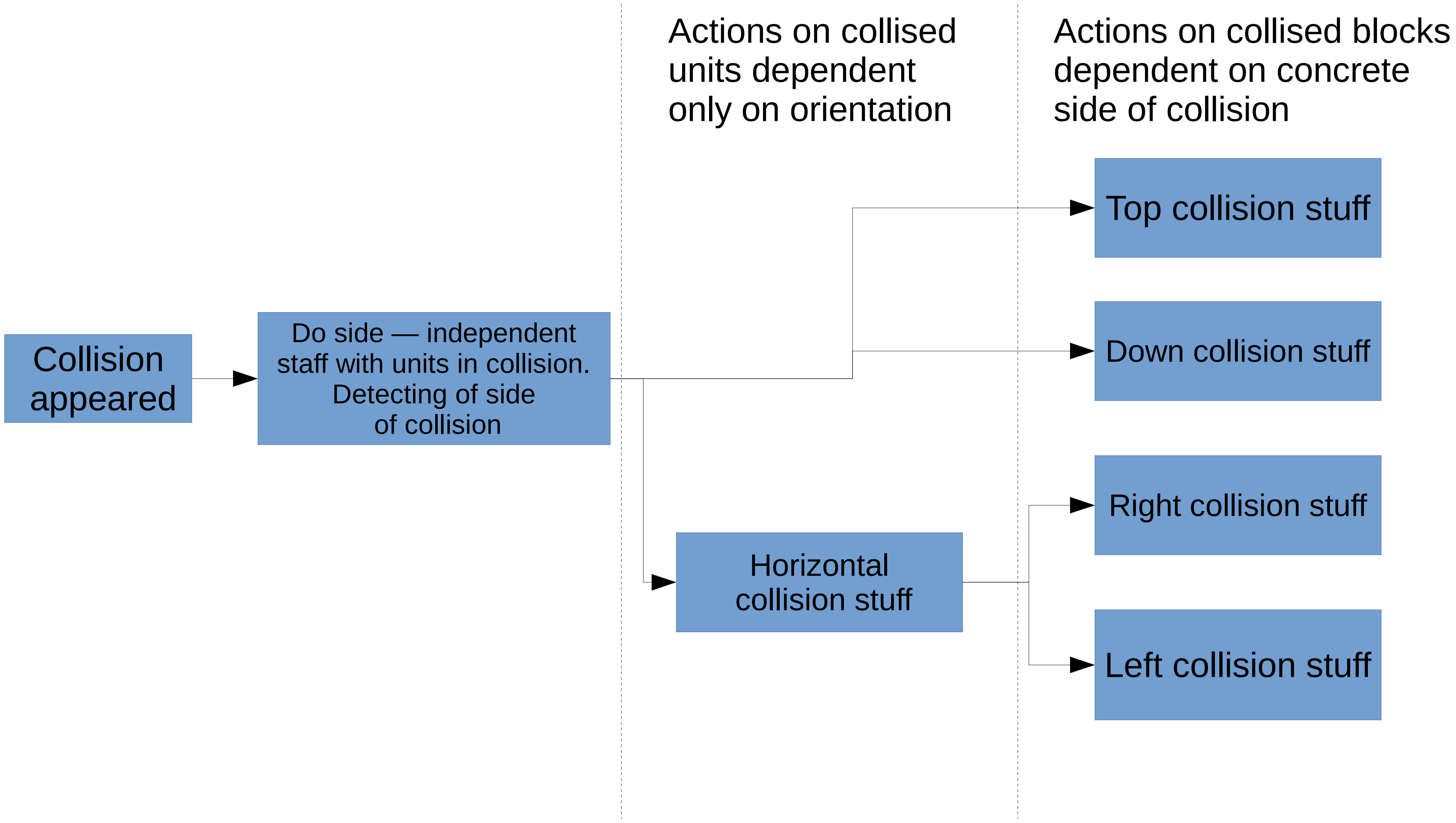
Detecting of the side of collision:

- let A unit has greater priority than B.
- we decide which side of A is bumped by B by angle between CA-CB line and horizontal axis.
- in our program angles start from -180° in the II part to 180° in the III part.





Collision events


Actions on collised units dependent only on orientation
Actions on collised blocks dependent on concrete side of collision
Collision appeared
Do side — independent staff with units in collision. Detecting of side of collision
Top collision stuff
Down collision stuff
Horizontal collision stuff
Right collision stuff
Left collision stuff

Collision events

- When collision appears world detects the side of it (e.g. unit A bumped to unit B by it`s left side, or if unit A felt down to grount it is collision between bottom of A ant top of ground unit)
- In our implementation in functions calculatiog side or orientation dependent staff unit A is unit with no lower priority then priority of unit B. It means that when player stays at ground, in collision functions player`s unit will be B and ground unit will be A. So this collision will be calculated by function toTop(ground, player) instead of toDown(player, ground).