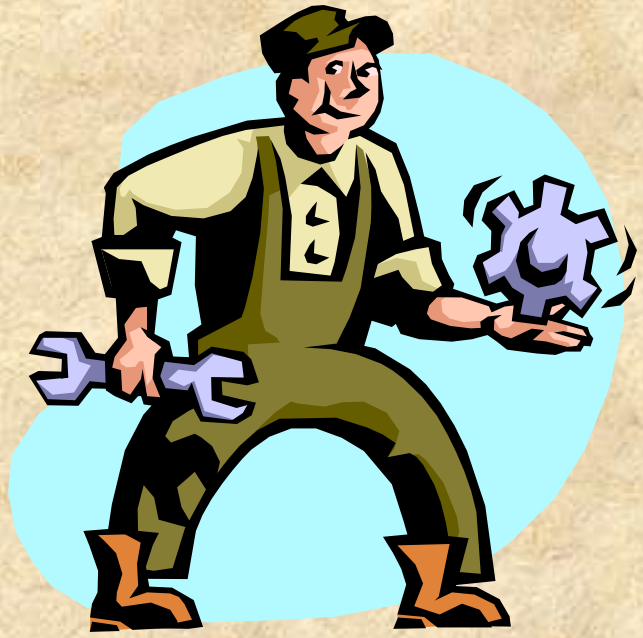
HY454 : ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ ΠΑΙΧΝΙΔΙΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,

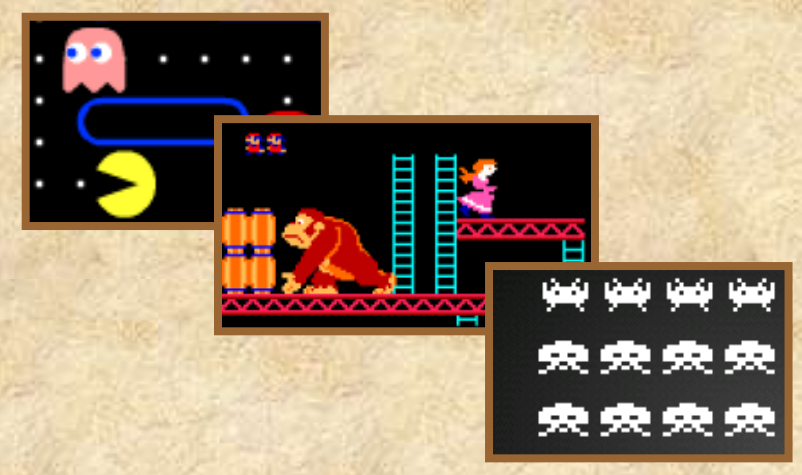
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ



ΔΙΔΑΣΚΩΝ

Αντώνιος Σαββίδης



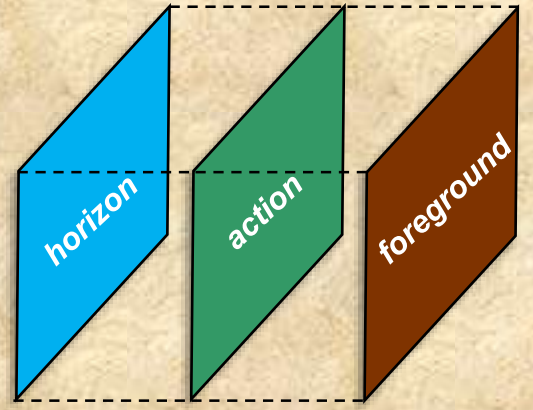
ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΩΝ,
Διάλεξη 6η

Περιεχόμενα

- *Multiple layers*
- Implementation
- Constrained motion

Multiple layers (1/11)

- Στα πραγματικά παιχνίδια εκτός από το χώρο της δράσης υπάρχουν και εναλλακτικά διαφορετικά terrains τα οποία παίζουν το ρόλο background και foreground
- Μία συνηθισμένη περίπτωση θα δούμε κατά την οποία θέλουμε πίσω από το βασικό terrain να φαίνεται ένας ορίζοντας ενώ από μπροστά θέλουμε να φαίνονται κάποια τμήματα τα οποία είναι μπροστά από το χώρο δράσης



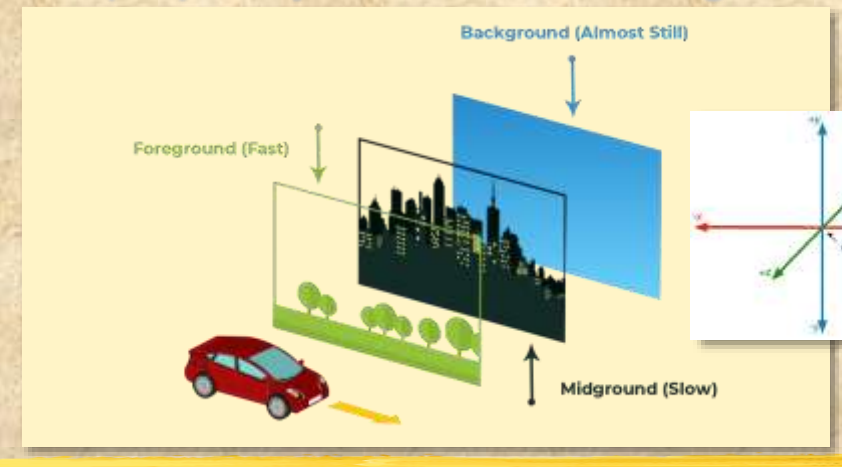
***Horizon*. Παίζει το ρόλο ενός background το οποίο κάνει scroll αρκετά αργά, συνήθως μόνο προς μία κατεύθυνση (π.χ. οριζόντια)**

***Action*. Είναι το βασικό layer (terrain) πάνω στο οποίο «κινούνται» και «ζουν» οι χαρακτήρες του παιχνιδιού**

***Foreground*. Είναι ένα συμπλήρωμα του action layer με σκοπό να προσθέσει σκηνικό το οποίο πρέπει να βρίσκεται μπροστά από το χώρο της δράσης**

Multiple layers (2/11)

- Τα layers ζωγραφίζονται με σειρά ***back → front***
- Έτσι τα back layers δημιουργούν την ψευδαίσθηση ότι είναι σε μεγαλύτερο βάθος (*low Z*) σε σχέση με τα front layers (*high Z*)
- Επίσης στα layers εφαρμόζεται scrolling με επιβραδυνόμενο ρυθμό *front → back*
- δηλαδή ενισχύεται η αίσθηση του βάθους κατά την κίνηση – αυτό ονομάζεται ***parallax scrolling***



Multiple layers (3/11)

- *Horizon layer (1/7)*
  - Tile-based terrain με διαστάσεις ανεξάρτητες του action
    - Μπορεί η οπτική να είναι πλήρως κατασκευασμένη με tiles
      - δύσκολο αν έχετε οπτικά περίπλοκο ορίζοντα
    - Αλλιώς δομείται οπτικά με game objects (δέντρα, λόφοι, σύννεφα, κλπ) τοποθετημένα σε κατάλληλα σημεία
      - Ίσως να χρειαστεί να έχετε και animations, όχι πάντα απαραίτητο
  - Heuristic background, που σημαίνει ότι υιοθετείτε κάποιον δικό σας τρόπο ειδικά για το background της σκηνής
    - δε συνηθίζεται πλέον εκτός και ένα έχετε πολύ απλό ορίζοντα
  - και στις δύο περιπτώσεις ο τρόπος με τον οποίο κάνει scroll εξαρτάται κυρίως από τη σχέση μεγέθους με το terrain layer
    - *perspective proportional scrolling (με προοπτική)*
    - *circular scrolling (με επανάληψη)*

Multiple layers (4/11)

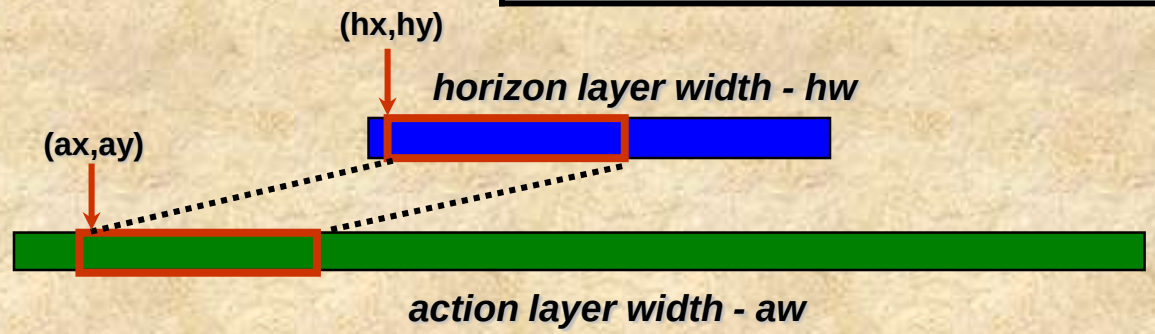
Horizon layer (2/7)

- Οι διαστάσεις των view και display windows είναι πάντα ίδιες για τα διαφορετικά layers
- Τα τμήματα που εκτυπώνονται στην οθόνη είναι αυτά των view windows.
- Το horizon layer είναι αρκετά μικρότερο σε διαστάσεις του action layer.

Στο perspective scrolling η σχετική θέση του view window στο horizon layer πρέπει να είναι ίδια με αυτή του αντίστοιχου view window στο action layer. Αυτό για την οριζόντια διάσταση υπολογίζεται ως:

$$hx = (hw-vw) * ax / (aw-vw)$$

Στο circular scrolling το horizon layer κάνει scroll ακριβώς με τον ίδιο ρυθμό και βήμα όπως το action layer, με τη διαφορά ότι μόλις δεν μπορεί να κάνει scrolling, π.χ. δεξιά, αυτομάτως το view window κάνει flip στην αριστερή πλευρά του layer. Αυτό δεν μπορεί όμως να εφαρμοστεί χωρίς προϋποθέσεις.



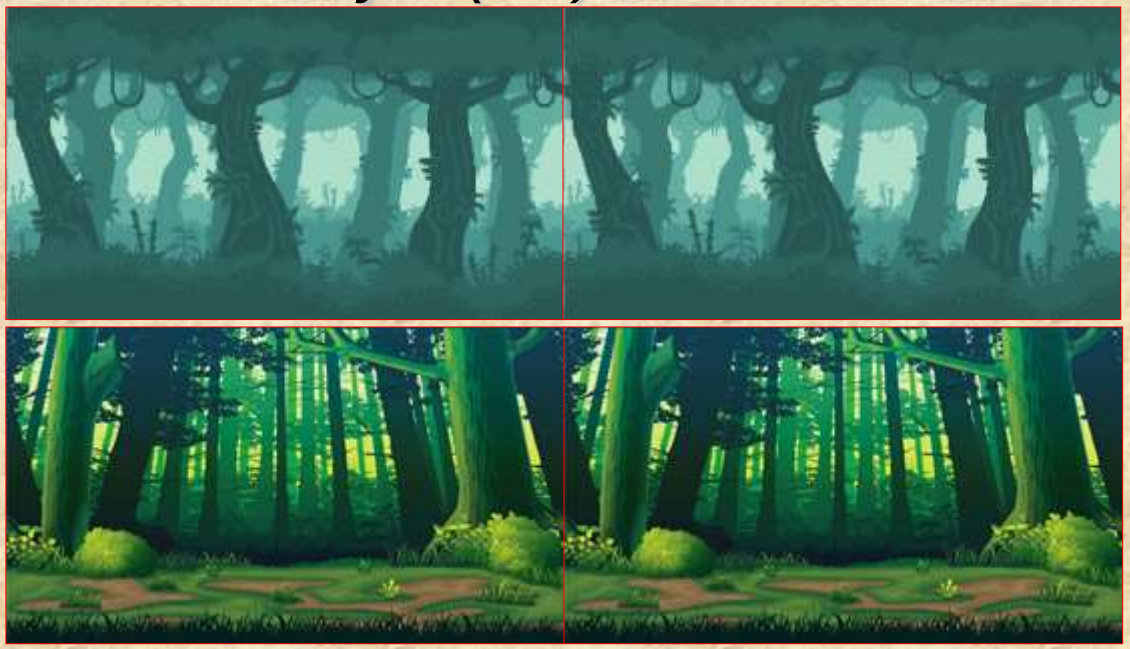
perspective scrolling

Multiple layers (5/11)

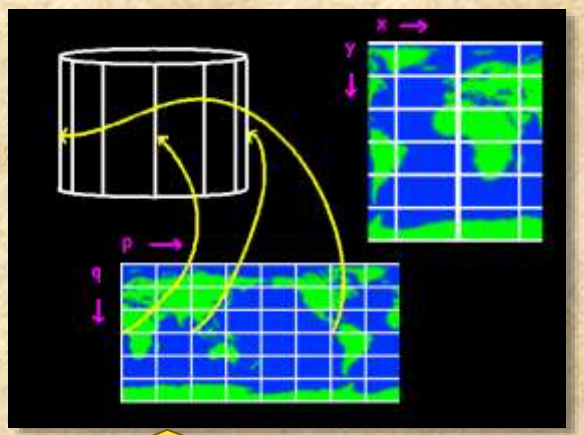
- *Horizon layer (3/7)*
  - Για να γίνει circular scrolling πρέπει η δομή του horizon layer να είναι οριζόντια / κάθετα κυκλική
  - για να έχει αυτό καλά οπτικά αποτελέσματα και να μη φαίνεται στατικό σχεδόν το background, θα πρέπει:
  - ***bitmap_width = view_width + C***
  - ***C*** → εύρος του scrolling distance - καλό είναι να ξεπερνά το μισό του ***bitmap_width***
    - ***bitmap_width >= view_width + ½ bitmap_width***
    - ***½ bitmap_width >= view_width***
    - *view window μικρότερο από το ήμισυ του background bitmap*

Multiple layers (6/11)

- *Horizon layer (4/7)*



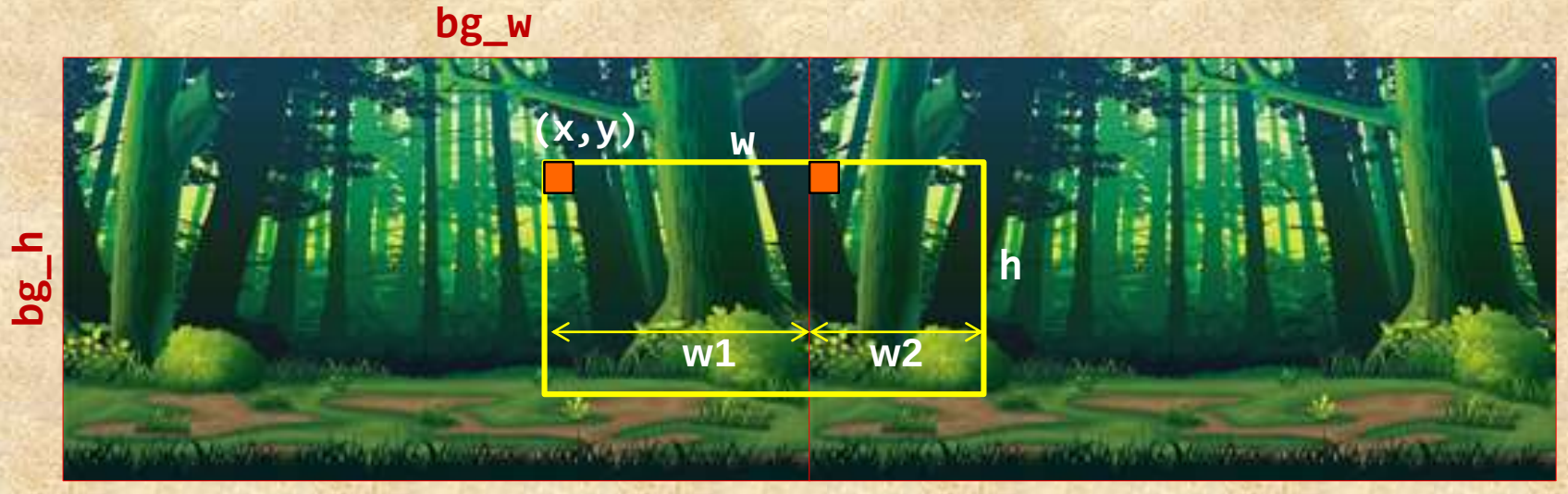
Sometimes we scroll the background in fast pace to give the illusion of speed in the foreground, while making it difficult to observe similarities



Rotating circular bitmaps, like an atlas, but visually making them to be more difficult to observe

Multiple layers (7/11)

- *Horizon layer (5/7)*



Multiple layers (8/11)

- *Horizon layer (6/7)*

```
class CircularBackground { // horizontal stripe
   private:
   Rect    viewWin;
   Bitmap  bg = nullptr;
   public:
   void    Scroll (int dx) {
             viewWin.x += dx;
             if (viewWin.x < 0)
                viewWin.x = BitmapGetWidth(bg) + viewWin.x;
             else
             if (viewWin.x >= BitmapGetWidth(bg))
                viewWin.x = viewWin.x - BitmapGetWidth(bg);
           }
   void    Display (Bitmap dest, int x, int y) const {
              auto bg_w = BitmapGetWidth(bg);
              auto w1   = std::min(bg_w - viewWin.x, viewWin.w);
              BitmapBlit(bg, { viewWin.x, viewWin.y, w1, viewWin.h }, dest, { x, y });
              if (w1 < viewWin.w) { // not whole view win fits
                 auto w2 = viewWin.w - w1; // the remaining part
                 BitmapBlit(bg, { 0, viewWin.y, w2, viewWin.h }, dest, { x + w1, y });
              }
           }
};
```

Multiple layers (9/11)

- *Horizon layer (7/7)*
  - Σε υλοποίηση ως tile-based terrain ισχύουν ότι έχουμε πει για το terrain ως προς το display και scrolling
    - Θέλει απλώς προσεκτική σχεδίαση των tiles ώστε να είναι όσο το δυνατόν λιγότερα
    - Μπορούν εύκολα να εφαρμοστούν και οι δύο περιπτώσεις scrolling
  - Σε περίπτωση bitmap για το horizon layer, πιο συνηθισμένη είναι η περίπτωση του circular scrolling
    - καθώς με perspective απαιτείται μεγάλο σχετικά bitmap για να φαίνεται κάποιου είδους scrolling στον ορίζοντα
    - μπορεί όμως να γίνει συνδυασμός tile-based horizon, χωρίς visual tiles, αλλά με moving objects με σταθερή κίνηση (π.χ. σύννεφα, πτηνά) ώστε να φαίνεται πιο ρεαλιστικός ο ορίζοντας

Multiple layers (10/11)

- *Foreground layer (1/2)*
  - Είναι πάντα tiled-based
  - Μπορεί να περιλαμβάνει στατικά objects που δεν αναπαρίστανται οπτικά με καλό τρόπο μέσω tiles
  - Έχει ακριβώς τις διαστάσεις του action layer
  - Υπόκειται αυτόματα μόνο στο *scrolling (dx,dy)* που εφαρμόζεται και στο action layer
    - ***Foreground synced to action***
  - Συνήθως το tile map για το foreground layer έχει μεγάλο αριθμό από empty tiles
  - Δε χρειάζεται συνοδευτικό grid
    - ***Foreground only serves visuals***

Multiple layers (11/11)

- *Foreground layer (2/2)*

Το rendering sequence είναι *back→front*, δημιουργώντας σε συνδυασμό με το parallax scrolling την αίσθηση βάθους

Περιεχόμενα

- Multiple layers
- *Implementation*
- Constrained motion

Implementation (1/4)

- `TileLayer` class
- `GridLayer` class
- Grid computation is separate
  - development time

```
class TileLayer {
private:
   Index*              map         = nullptr;
   GridLayer*          grid        = nullptr;
   Dim                 totalRows = 0, totalColumns = 0;
   Bitmap              tileSet     = nullptr;
   Rect                viewWin;
   Bitmap              dpyBuffer   = nullptr;
   bool                dpyChanged = true; ← this has changed to cache VW
   Dim                 dpyX = 0, dpyY = 0;
   void                Allocate (void) {
                          map = new Index [totalRows * totalColumns];
                          dpyBuffer = BitmapCreate(
                             GetResWidth()   + 2 * TILE_WIDTH,
                             GetResHeight()  + 2 * TILE_HEIGHT
                          );
                       }
```

Keep an extra tile rows and tile columns just in case screen is not divisible by their dimensions

Implementation (2/4)

```
public:
void                SetTile (Dim col, Dim row, Index index);
Index               GetTile (Dim col, Dim row) const
                              { return map[row * totalColumns + col]; }
const Point         Pick (Dim x, Dim y) const {
                       return { DIV_TILE_WIDTH(x + viewWin.x),
                                DIV_TILE_HEIGHT(y + viewWin.y) };
                    }
const Rect&         GetViewWindow (void) const { return viewWin; }
void                SetViewWindow (const Rect& r)
                              { viewWin = r; dpyChanged = true; }
void                Display (Bitmap dest, const Rect& displayArea);

Bitmap              GetBitmap (void) const { return dpyBuffer; }
int                 GetPixelWidth (void) const { return viewWin.w; }
int                 GetPixelHeight (void) const { return viewWin.h; }
unsigned            GetTileWidth (void) const { return DIV_TILE_WIDTH(viewWin.w); }
unsigned            GetTileHeight (void) const { return DIV_TILE_HEIGHT(viewWin.h); }

void                Scroll (float dx, float dy);
bool                CanScrollHoriz (float dx) const;
bool                CanScrollVert (float dy) const;
```

Implementation (3/4)

```
   auto                ToString (void) const -> const std::string; // unparse
   bool                FromString (const std::string&); // parse
   void                Save (const std::string& path) const
                         { fclose(WriteText(fopen(path.c_str(), "wt"))); }

   bool                Load (const std::string& path);
   FILE*               WriteText (FILE* fp) const
                          { fprintf(fp, "%s", ToString().c_str()); return fp; }
   bool                ReadText (FILE* fp); // TODO: carefull generic parsing

   TileLayer (Dim _rows, Dim _cols, Bitmap _tileSet);
   ~TileLayer (); // cleanup here with care!
};
```

Implementation (4/4)

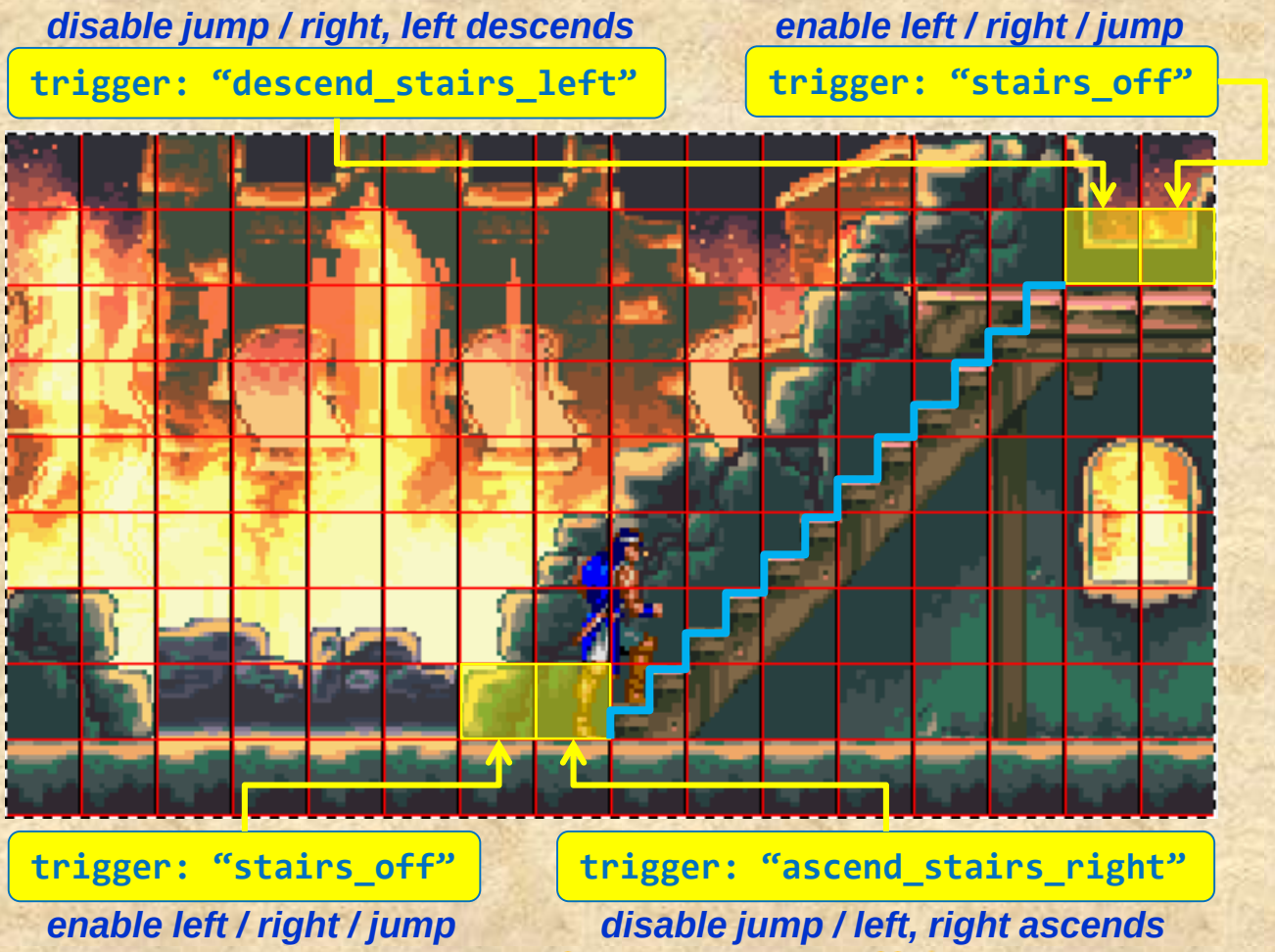
```
class GridLayer {
   private:
   GridIndex*          grid = nullptr;
   unsigned            total = 0;
   Dim                 totalRows = 0, totalColumns = 0;
   void                Allocate (void) {
                          grid = new GridIndex [total = totalRows * totalColumns];
                          memset(grid, GRID_EMPTY_TILE, total);
                       }
   // TODO: adapt as needed and insert all rest motion control functions
   // inside the private section
   void                FilterGridMotionDown (const Rect& r, int* dy) const;

   public:
   void                FilterGridMotion (const Rect& r, int* dx, int* dy) const;
   bool                IsOnSolidGround (const Rect& r) const { // will need later for gravity
                         int dy = 1;          // down 1 pixel
                         FilterGridMotionDown(r, &dy);
                         return dy == 0;      // if true IS attached to solid ground
                       }
   GridIndex*&         GetBuffer (void) { return grid; }
   const GridIndex*&   GetBuffer (void) const { return grid; }
   GridLayer (unsigned rows, unsigned cols);
};
```

Constrained motion (1/5)

- Είναι ο περιορισμός κινήσεων του player σε συγκεκριμένες ζώνες του terrain
- Έτσι απλοποιείται η υλοποίηση της κίνησης καθώς και οι υπολογισμοί που απαιτούνται
- Συνήθως σε ***stairs***, ***tubes*** (predefined paths), ***slopes***, any ***custom surfaces***
- Γίνεται με δύο τρόπους:
  - custom ***collision shapes*** (triggers / landmarks)
  - ***tagged tiles*** with cross notifications

Constrained motion (2/5)



Constrained motion (3/5)

```
// tiles may be given a type tag associated with
// a game action to be performed upon CROSSING it

class TileLayer;
class TileActions {
   public:
   using Action       = std::function<void(Dim col, Dim row)>;

                      // (row,col)->unique tile number (in terrain map)
   using Enumerator   = std::function<unsigned(Dim col, Dim row)>;

   private:
   using Actions      = std::map<std::string, Action>;
   using Tags         = std::map<std::string, std::set<unsigned>>;
   Actions            actions;
   Tags               tags;
   Enumerator         enumerator;
   static TileActions singleton;
   …
```

Constrained motion (4/5)

```
   public:
   template <typename Tfunc>
   void    SetEnumerator (const Tfunc& f)
              { enumerator = f; }
   template <typename Tfunc>
   void    Install (const std::string& tag, const Tfunc& f)
              { actions[tag] = f; }
   void    SetTag (Dim col, Dim row, const std::string& tag)
              { tags[tag].insert(enumerator(col, row)); }

   void    Trigger (Dim col, Dim row) {
              auto pos = enumerator(col, row);
              for (auto& i : tags)
                     if (i.second.find(pos) != i.second.end()) {
                                auto j = actions.find(i.first);
                                if (j != actions.end())
                                     j->second(col, row);
                                return;
                     }
           }
   static auto GetSingleton (void) -> TileActions&
                     { return singleton; }
   static auto GetSingletonConst (void) -> const TileActions&
                     { return singleton; }
};
```

Constrained motion (5/5)

```
void FilterGridMotionLeft (GridMap* m, const Rect& r, int* dx, std::list<Point>* crossedTiles) {
          …as before ..
         std::list<Point> l;
          for (auto row = startRow; row <= endRow; ++row)
             if (!CanPassGridTile(m, newCol, row, GRID_RIGHT_SOLID_MASK)) {
                    *dx = r.x - MUL_GRID_ELEMENT_WIDTH(currCol);
                    l.clear();
                    break;
             }
             else
                    l.push_back({newCol, row});
         std::copy(l.begin(), l.end(), std::back_inserter(*crossedTiles));
}

void FilterGridMotion (GridMap* m, const Rect& r, int* dx, int* dy) {
         std::list<Point> crossedTiles;
          …as before, with &crossTiles as extra last argument to all Filter methods.
         for (auto& i : crossedTiles)
             TileActions::GetSingleton().Trigger(i.x, i.y);
}
```