

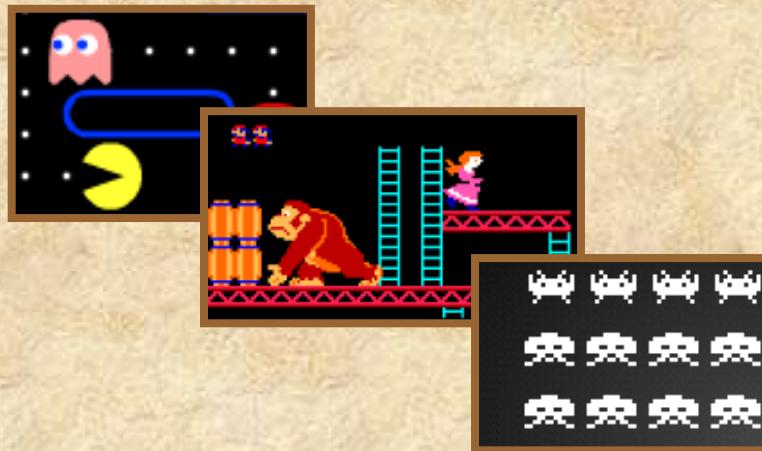


**HY454 : ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ
ΠΑΙΧΝΙΔΙΩΝ**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ**



**ΔΙΔΑΣΚΟΝΤΕΣ
Αντώνιος Σαββίδης**



**ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΩΝ,
Διάλεξη 7η**

Key frame animation



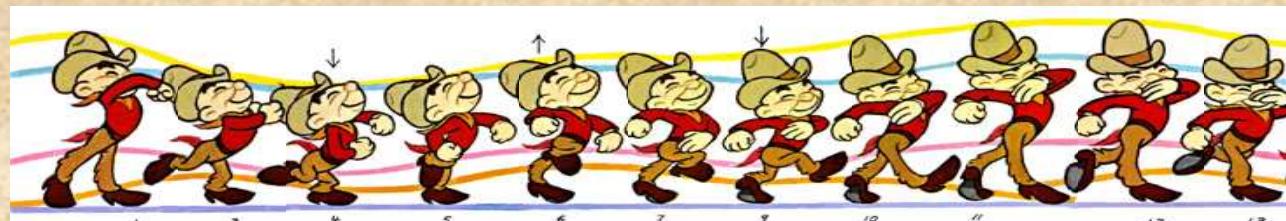
Περιεχόμενα

- ***Key frame animation***
- Animation films



Key frame animation (1/4)

- Η πιο απλή περίπτωση animation είναι αυτή που υπάρχει από την εποχή των πρώτων φιλμ «κόμικς»
 - βασίζεται στην **αλληλουχία εικόνων** οι οποίες δείχνουν διαδοχικές χρονικά απόψεις της σκηνής και των χαρακτήρων
- Στις ταινίες αυτές οι ενέργειες των βασικών ηρώων αποτυπώνονταν σε ξεχωριστές διαφάνειες, μία για κάθε διαδοχική κίνηση
 - ενώ παρουσιάζονταν με στιγμιαίες λήψεις όπου πάνω στη βασική σκηνή (**background**) τοποθετούνταν μία διαφάνεια κάθε φορά (**key frame**)



From the book of Preston Blair, "Cartoon Animation",
<http://www.freetoon.com/prestonblair/intro/main.html>





Key frame animation (2/4)

- Η κάθε σκηνή έχει το δικό της χρονισμό ενώ η αλληλουχία των σκηνών με τα σημεία κλειδιά της δράσης υπάρχει αποτυπωμένη σε ειδικό story board



- Ο αριθμός των στιγμιαίων λήψεων (snapshots) μπορεί να είναι μεγαλύτερος από τον αριθμό των διαφανειών ώστε να επιτυγχάνεται ο σωστός χρονισμός
- Αυτό γίνεται γνωρίζοντας την ταχύτητα προβολής του «φιλμ» που είναι σταθερή, δηλ. γνωρίζοντας ακριβώς τον χρόνο μεταξύ δύο διαδοχικών «καρέ» της ταινίας.

- Τα συγκεκριμένα frames που αποτυπώνονται στις διαφάνειες ονομάζονται key frames, καθώς αποτελούν επιλεγμένες ενδιάμεσες περιπτώσεις της αναλογικής κίνησης των χαρακτήρων.
- Ο τρόπος δημιουργίας των ταινιών αυτών έχει υιοθετηθεί σχεδόν εξ ολοκλήρου και στα παιχνίδια, καθώς στην πράξη τείνουν να είναι interactive movies.



Key frame animation (3/4)

- Στα 2d games η τακτική αυτή εφαρμόζεται σχεδόν πανομοιότυπα, ενώ υλοποιείται και με τον ίδιο τρόπο, καθώς σήμερα στις ταινίες cartoon όλοι οι χαρακτήρες σχεδιάζονται και αποτυπώνονται ψηφιακά
 - δεν υπάρχουν διαφάνειες ούτε κάμερες λήψης, αλλά software το οποίο αποτυπώνει όλους του χαρακτήρες στις θέσεις τους σε κάθε σκηνή
 - ενώ στο τέλος όλες οι ψηφιακά αποτυπωμένες σκηνές εμφανίζονται σε κανονικό κινηματογραφικό «φιλμ»
- Η εκτύπωση των χαρακτήρων γίνεται πάντοτε με **MaskedBlit**, ενώ οι διαδοχικές αποτυπώσεις γίνονται με χρονισμό που ελέγχεται από το software (π.χ. 25 msec για κάθε key frame), ανάλογα βέβαια με το είδος και την ταχύτητα της κίνησης
- Εάν έχουμε N διαφορετικά bitmaps ενός χαρακτήρα τα οποία αποτυπώνουν κάποια κίνηση και γνωρίζουμε τον χρονισμό t και τις θέσεις τους p_1, \dots, p_n στη βασική σκηνή, μπορούμε να τα αποτυπώσουμε εύκολα με ένα loop →



Key frame animation (4/4)

- Ο απλούστερος τρόπος υλοποίησης ενός non-interactive key frame animation για έναν και μόνο χαρακτήρα, όπως θα αποτυπώνονταν σε ένα cartoon movie

```
#define TOTAL_KEY_FRAMES 10
#define FRAME_DELAY 50
#define BLACK_COLOR 0

Bitmap keyFrames[TOTAL_KEY_FRAMES];
Point framePositions[TOTAL_KEY_FRAMES];
void Animate (void) {
    uint64_t t = 0;
    for (auto i = 0; i < TOTAL_KEY_FRAMES; )
        if (CurrTime() >= t) {
            t = CurrTime() + FRAME_DELAY;
            auto b = keyFrames[i];
            Vsync();
            BitmapClear(BitmapGetScreen(), BLACK_COLOR);
            MaskedBlit(
                b,
                { 0, 0, BitmapGetWidth(b), BitmapGetHeight(b) },
                BitmapGetScreen(),
                framePositions[i++]
            );
        }
}
```

positioning

timing

masking

frame change



Περιεχόμενα

- Key frame animation
- *Animation films*



Animation films (1/13)

- **Animation film** (ή **sprite sheet** παραδοσιακά) λέγεται ένα bitmap το οποίο περιέχει όλα τα key frames για μία ή περισσότερες ενέργειες ενός χαρακτήρα του παιχνιδιού
- Η τοποθέτηση των key frames μέσα στο animation film δεν ακολουθεί κάποια προκαθορισμένη τοπολογία, αλλά ο σκοπός είναι πάντα η εύκολη πρόσβαση (πρακτικό θέμα)
- Αποφεύγουμε να έχουμε συνήθως πολλές διαφορετικές κινήσεις μέσα στο ίδιο animation film κυρίως για λόγους ευκολίας επεξεργασίας από τους γραφίστες
- Αποφεύγουμε να έχουμε κινήσεις που αφορούν διαφορετικούς χαρακτήρες μέσα στο ίδιο animation film, καθώς αυτό δημιουργεί προβλήματα στην σωστή οργάνωση και ονομασία των αρχείων - films (π.χ. ανά όνομα χαρακτήρα), ενώ εμποδίζει ενδεχομένως τακτικές όπως *loading on demand*

Animation films (2/13)

- Συνηθίζεται τα διαδοχικά frames να τοποθετούνται ομοιόμορφα (δηλ. το καθένα να καταλαμβάνει χώρο ενός ορθογώνιου ιδίων διαστάσεων με τα υπόλοιπα)
- Καθώς και να τοποθετούνται με κάποια φυσική αλληλουχία (π.χ. από αριστερά προς τα δεξιά , από πάνω προς τα κάτω)
- Ωστόσο, μερικές φορές φτιάχνονται εργαλεία που κάνουν βελτιστοποίηση χώρου ενός film και τοποθετούν τα frames σε διάφορες θέσεις ώστε να έχουμε το μικρότερο δυνατό film bitmap



4x7,W:88,H:102

Ένθετο

- Στο παρελθόν για εξοικονόμηση χώρου συνηθίζονταν διαφορετικοί χαρακτήρες κατ τα frames τους ή οι κινήσεις τους να «συμπιέζονταν» στο ίδιο film (bitmap)
 - Αυτό ισχύει και σήμερα αλλά για λόγους ταχύτητας και όχι για λόγους μνήμης





Animation films (3/13)

- Δεν είναι απαραίτητο τα αυθεντικά films να περιέχουν τα frames σε minimal bounding boxes (*non-uniform frames*)
 - Αυτό ωστόσο μπορεί να γίνει από εργαλείο που κάνει βελτιστοποίηση χώρου
 - Μπορούμε να έχουμε τα *original films* (development time) και τα *compacted films* (run time) ή αλλιώς ***sprite sheets***
 - ◆ Αν και αρκετοί προτιμούν να χρησιμοποιούν απευθείας τα original films εάν δεν συντρέχουν σοβαρά προβλήματα χώρου (είτε στο δίσκο ή στη μνήμη) και ταχύτητας
 - Είτε χρησιμοποιείτε τα original films είτε τα compacted films, πρέπει να έχετε πάντα πληροφορία για τα rectangles του κάθε frame κάπου
 - → ***uniform frames make your life easier***

Animation films (4/13)

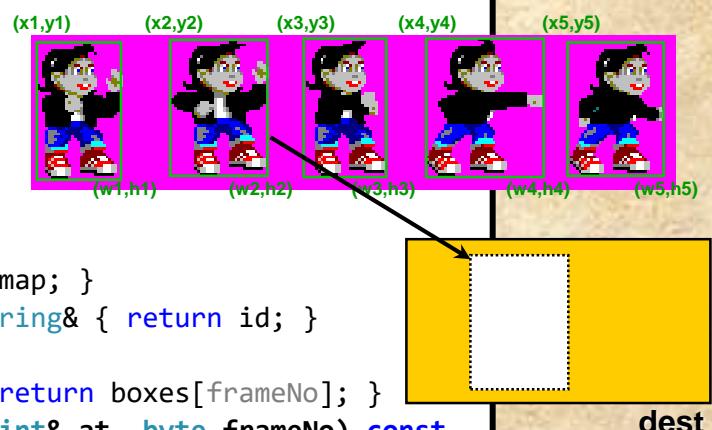
- Τα οριζόντια films, με ένα animation ανά film, είναι μία καλή και επαρκής τακτική
- Ο χώρος του κάθε frame προκύπτει πολύ εύκολα, αλλά αυτό δεν συνιστά το minimal bound box
- Η προ-επεξεργασία του film πρέπει παράγει τη λίστα των minimal bounding boxes για κάθε frame
- Αυτή η λίστα πρέπει να «συνοδεύει» κάθε film καθώς αυτή προσδιορίζει τον ακριβή χώρο του κάθε frame



Animation films (5/13)

- Τα animation films είναι instances της αντίστοιχης κλάσης. Όπως φαίνεται, το bitmap και η λίστα των bounding boxes είναι *ctor arguments*, καθώς θέλουμε να είμαστε ευέλικτοι ως προς τον τρόπο που γίνεται το *loading*

```
class AnimationFilm {  
    std::vector<Rect>    boxes;  
    Bitmap                bitmap = nullptr;  
    std::string            id;  
public:  
    byte                 GetTotalFrames (void) const  
                        { return boxes.size(); }  
    Bitmap               GetBitmap (void) const { return bitmap; }  
    string               GetId (void) const -> const std::string& { return id; }  
    Rect&               GetFrameBox (byte frameNo) const  
                        { assert(boxes.size() > frameNo); return boxes[frameNo]; }  
    void                 DisplayFrame (Bitmap dest, const Point& at, byte frameNo) const  
                        { MaskedBlit(bitmap, GetFrameBox(frameNo), dest, at); }  
    void                 SetBitmap (Bitmap b)  
                        { assert(!bitmap); bitmap = b; }  
    void                 Append (const Rect& r) { boxes.push_back(r); }  
    AnimationFilm (const std::string& _id) : id (_id){}  
    AnimationFilm (Bitmap, const std::vector<Rect>&, const std::string&);  
};
```





Animation films (6/13)

```
void Animate (const AnimationFilm& film, const Point& at) {  
    uint64_t t = 0;  
    for (byte i = 0; i < film.GetTotalFrames(); )  
        if (CurrTime() >= t) {  
            t = CurrTime() + FRAME_DELAY;  
            Vsync();  
            BitmapClear(BitmapGetScreen(), BLACK_COLOR);  
            film.DisplayFrame(BitmapGetScreen(), at, i++);  
        }  
}
```

Η πολύ απλή Animate προσαρμόζεται στη χρήση των animation films. Άλλα και πάλι να θυμάστε πως ποτέ δεν κάνουμε *animate* με αυτό τον τρόπο – πρόκειται απλώς για διδακτικό παράδειγμα.

- Η πληροφορία για τα διάφορα animation films ορίζεται σε κάποιο data file, ενώ μπορώ να έχω διαφορετικά τέτοια files ανά χαρακτήρα αλλά και επίπεδο του παιχνιδιού
- Αυτό σημαίνει ότι τα films φορτώνονται μαζικά και «κρατούνται» σε κάποιο singleton (holder class) από το οποίο ζητούνται βάσει id



Animation films (7/13)

Sample catalogue file with all films:

```
"player.punch" "films/player/shoot.bmp"
3
5 5    34 60
5 35   28 67
7 50   20 58
```

Use "\$" as film id to designate end

Respective catalogue grammar:

```
Catalogue  : Film+ "$"

Film       : Id           <quoted_string>
             Path         <quoted_string>
             TotalFrames <number>
             Rect+
             ...
             ...

Rectangle : X            <number>
             Y            <number>
             Width        <number>
             Height       <number>
```

Use JSON instead for all data and configuration

```
"films" : {
  "player.punch" :{
    "bmp" : "punch.bmp",
    "frames" : [
      { "x":5, "y":5,"w":34,"h":60 },
      ... rest frames
    ]
  },
  ... rest films
}
```



Animation films (8/13)

```
class AnimationFilmHolder final {
public:
    using Parser = std::function<
        bool (std::list<AnimationFilm::Data>& output, const std::string& input)
    >;
    using EntryParser = std::function<
        int (// -1=error, 0=ended gracefully, else #chars read
            int startPos,
            const std::string& input,
            std::string& idOutput,
            std::string& pathOutput,
            std::vector<Rect>& rectsOutput
        )
    >;
private:
    using Films = std::map<std::string, AnimationFilm*>;
    Films films;
    BitmapLoader bitmaps; // only for loading of film bitmaps
    static AnimationFilmHolder holder; // singleton

    AnimationFilmHolder (void) {}
    ~AnimationFilmHolder() { CleanUp(); }

    static auto Get (void) -> const AnimationFilmHolder& { return holder; }
    void Load (const std::string& text, const EntryParser& entryParser);
    void Load (const std::string& text, const Parser& parser);
    void CleanUp (void);
    auto GetFilm (const std::string& id) -> const AnimationFilm* const;
```



Animation films (9/13)

```
void AnimationFilmHolder::Load (const std::string& text, const EntryParser& entryParser) {
    int pos = 0;
    while (true) {
        std::string id, path;
        std::vector<Rect> rects;
        auto i = entryParser(pos, text, id, path, rects);
        assert(i >= 0);
        if (!i) return;
        pos += i;
        assert(!GetFilm(id));
        films[id] = new AnimationFilm(bitmap.Load(path), rects, id);
    }
}
void AnimationFilmHolder::Load (const std::string& text, const Parser& parser) {

    std::list<AnimationFilm::Data> output;
    auto result = parser(output, text);
    assert(result);

    for (auto& entry : output) {
        assert(!GetFilm(entry.id));
        films[entry.id] = new AnimationFilm(
            bitmap.Load(entry.path), entry.rects, entry.id
        );
    }
}
```



Animation films (10/13)

```
// .. continued
void CleanUp (void) {
    for (auto& i : films)
        delete(i.second);
    films.clear();
}

auto GetFilm (const std::string& id) -> const AnimationFilm* const {
    auto i = films.find(id);
    return i != films.end() ? i->second : nullptr;
}
};
```

- Η χρήση ενός κοινού singleton films holder για όλους τους χαρακτήρες απαιτεί prefixing των films με το character name
 - “player.jump”, “rabit.jump”, “antetokounmpo.jump”



Animation films (11/13)

- Μένει να δούμε «ποιος» (δηλ. κλάση ή τμήμα) φέρει την ευθύνη για o loading των bitmaps και την αποθήκευση αντιστοίχων των instances
 - Είδαμε ότι το animation film δέχεται το bitmap ως constructor argument
 - ενώ o film holder δεν ασχολείται παρά μόνο σε υψηλότερο επίπεδο με films
- Υπάρχει για το σκοπό αυτό o bitmap loader, o οποίος φροντίζει να φορτώνει ένα bitmap αλλά και να κρατάει όλα τα loaded instances σε εσωτερικές δομές
 - έτσι και το debugging γίνεται ευκολότερο, καθώς υπάρχουν όλα τα bitmaps συγκεντρωμένα σε μία holder class
 - και το clean-up ευκολότερο, καθώς μόνο οι holder class αρκεί να κάνει destroy τα bitmaps
 - και οι έλεγχοι πιο εξονυχιστικοί, καθώς μπορούμε να κάνουμε validate κάθε bitmap instance μέσω του bitmap loader



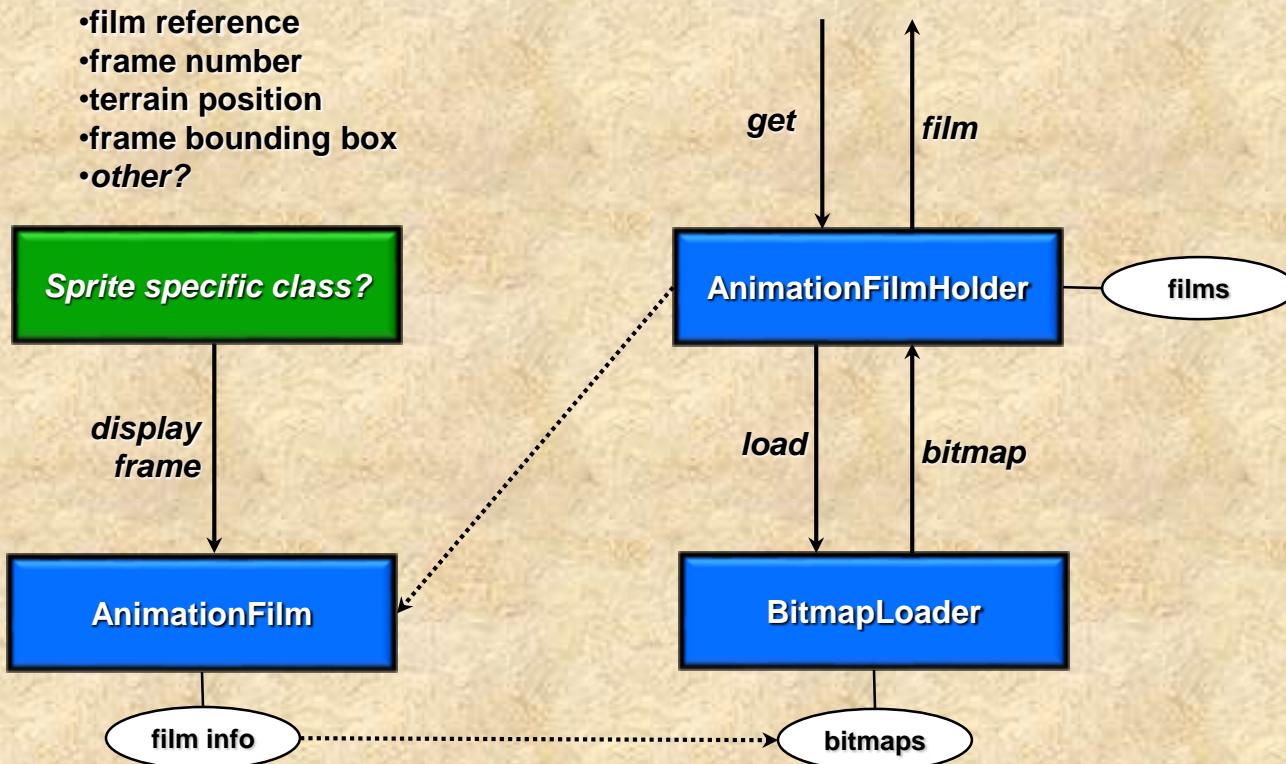
Animation films (12/13)

```
class BitmapLoader {  
private:  
    using Bitmaps = std::map<std::string, Bitmap>;  
    Bitmaps bitmaps;  
    Bitmap GetBitmap (const std::string& path) const {  
        auto i = bitmaps.find(path);  
        return i != bitmaps.end() ? i->second : nullptr;  
    }  
  
public:  
    Bitmap Load (const std::string& path) {  
        auto b = GetBitmap(path);  
        if (!b) {  
            bitmaps[path] = b = BitmapLoad(path);  
            assert(b);  
        }  
        return b;  
    }  
    // prefer to massively clear bitmaps at the end than  
    // to destroy individual bitmaps during gamePlay  
    void CleanUp (void) {  
        for (auto& i : bitmaps)  
            BitmapDestroy(i.second);  
        bitmaps.clear();  
    }  
    BitmapLoader (void){}  
    ~BitmapLoader() { CleanUp(); }  
};
```

- Με αυτό τον τρόπο τοποθετούμε την ευθύνη για bitmap loading αλλά και για bitmap destruction κεντρικά σε μία εύκολα ελεγχόμενη κλάση
- Μπορεί για το ίδιο το bitmap να έχουμε πολλές load κλήσεις χωρίς πρόβλημα πολλαπλών loadings
- Μπορούμε να έχουμε όσα bitmap instances (pointers) θέλουμε μέσα στον κώδικα και να μην ασχολούμαστε με το πότε θα πρέπει να κάνουμε destroy
- Μπορούμε να μετατρέψουμε τον loader σε singleton, εάν θέλουμε όλα τα bitmaps να είναι συγκεντρωμένα, ή να έχουμε διαφορετικούς ανά κατηγορία bitmaps
- Μπορούμε να προσθέσουμε reference counting και να κάνουμε destroy μόνο στο τελευταίο unload, αλλά δεν είναι τόσο καλή πρακτική.



Animation films (13/13)



Uniform vs minimal BBs (1/5)

An example animation film for *run* action



Unfortunately, the baselines of the frames that are supposed to touch the ground { 0, 1, 4, 5 } do not coincide each other, so successive rendering will show the character moving up and down while running

Correct baselines, uniform frames behave well





Uniform vs minimal BBs (2/5)

- Η χρήση των uniform BBs σημαίνει απλούς υπολογισμούς, πχ, στην περίπτωση των horizontal films:
 - `frame_width = film_width / total_frames`
 - `frame_height = film_height`
- Τότε οι κινήσεις σε επίπεδο (γραμμή) δεν πρέπει να απαιτούν extra Y *offsetting* per frame
 - όπως στο δεύτερο παράδειγμα πριν (well-aligned baselines)
- Ωστόσο, δεν γίνεται για όλων των ειδών τις κινήσεις να ενσωματωθεί το Y *offsetting* στα film
 - *jumps, slop ascending / descending, stair stepping*



Uniform vs minimal BBs (3/5)



jump sequence in three successive animations

Frames do not occupy the same horizontal space, thus sizes should accompany the film (JSON or .csv)

But heights ARE ALWAYS uniform even when widths differ! Never break that!

- Η παραπάνω τακτική εάν επιβάλλουμε επιπλέον και uniform width, είναι η καλύτερη για την δημιουργία films για animations που **κατά την εκτέλεση θα έχουν και Y offsetting**
 - Λόγω του uniform width (που είναι max frame width) χάνουμε σε ακρίβεια collision
 - Όμως τις περισσότερες φορές αυτό δε μας πειράζει

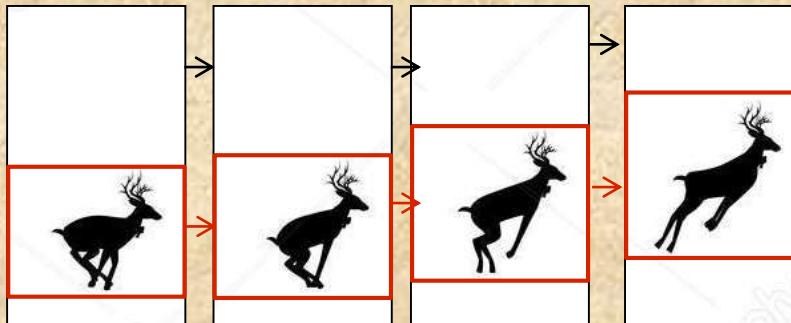
Uniform vs minimal BBs (4/5)



- Η παραπάνω τακτική αν και έχει uniform frames πάσχει λόγω του ότι τα animations έχουν **κατά την σχεδίαση Y offsetting**
 - Εδώ μπορεί να χάσουμε πολύ σε ακρίβεια collision
 - Ωστόσο εάν: (i) κάνουμε conditionally disable το collision checking κατά τη διάρκεια τέτοιων κινήσεων, και (ii) αυτό είναι αποδεκτό ως gameplay feature, τότε είναι η καλύτερη επιλογή

Uniform vs minimal BBs (5/5)

key frame transition, Y unchanged

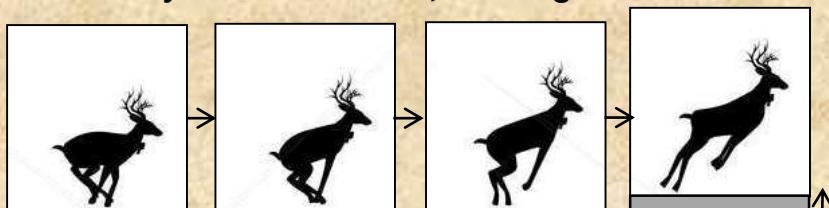


bbox transition, changed with frames

- **Pros:** no Y computations involved in animation as graphic artists encompass offsets animation in frames

- **Cons:** requires bbox updates during animation, and disables shifting from the original animation offsets if required

key frame transition, Y changes as needed



dy, changed with frames

- **Pros:** fixed bbox, (dx, dy) both naturally apply for character motion

- **Cons:** initial offsets must be taken from the original film made by the graphic artist (*well, pretty easy*)