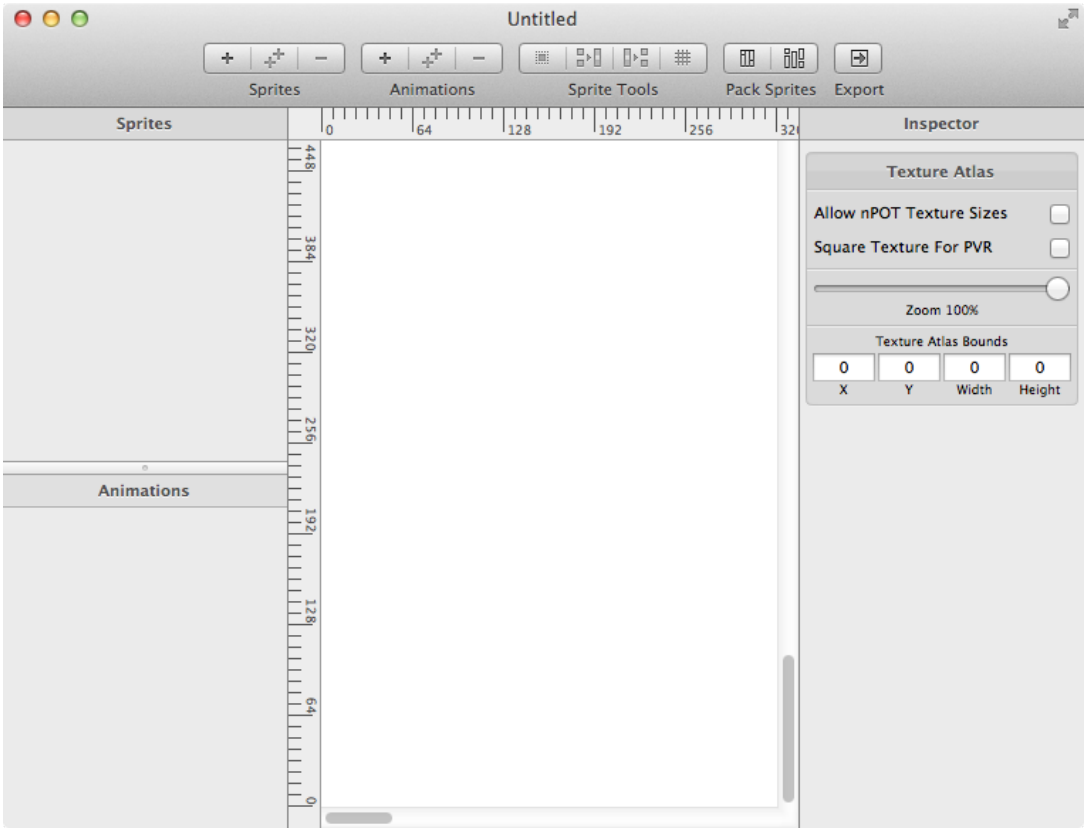
Atlas

Tutorial

Table of contents

1. Creating a new project

Create a new project by starting Atlas.app and clicking on the File > New... menu item. This will open a blank Atlas project, and show the Sprite Sheet Editor:



This screen consists of 5 parts:

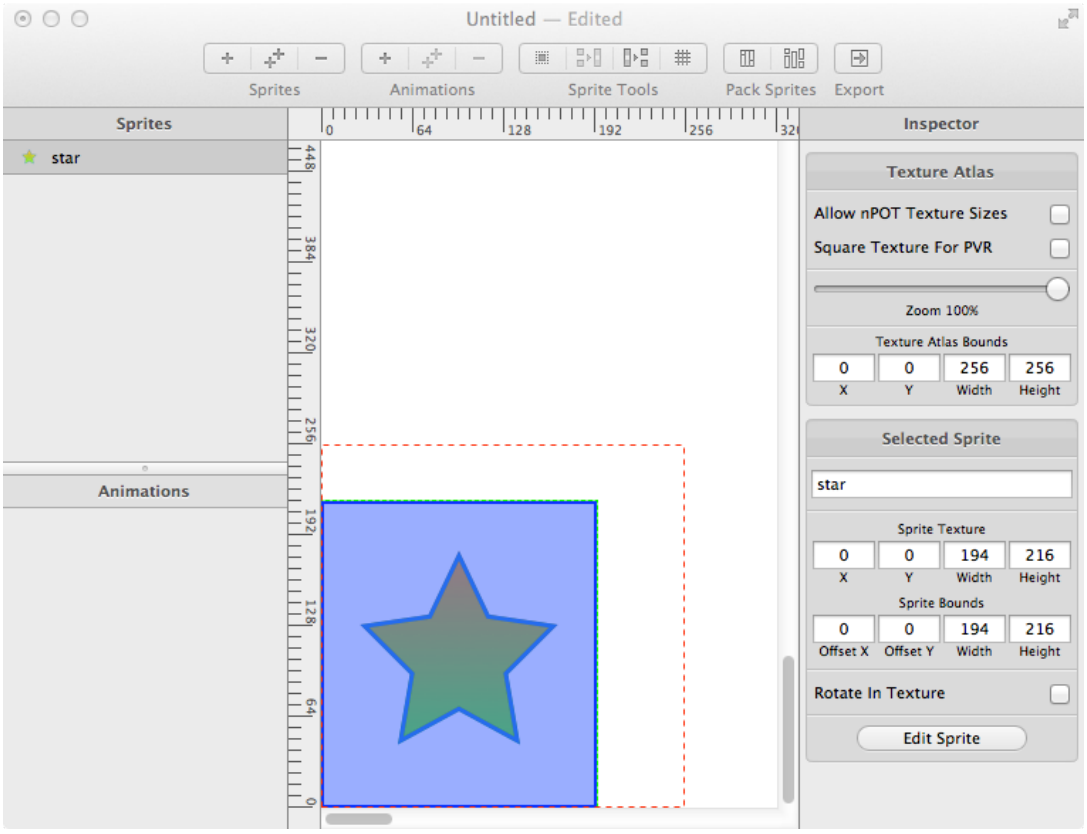
- the Toolbar - here You can find buttons for the most important commands
- the Sprites list - here you will see the list of Your sprites
- the Animations list - here you will see the list of Your animations
- the Editor - this will show a preview of the sprite sheet You are creating
- the Inspector - here You will find panels showing the settings for the editor and the selected objects

Later in this tutorial we will cover all aspects of these.

In this tutorial we will create a demo project, using most of the features in Atlas, and in the last part we will create a small demo in the cocos2d environment to show the strengths of Atlas.

2. Creating a sprite

Let's assume we have an image which we previously created using some graphics editor, called **star.png** (which can be downloaded from the Atlas home page). This is a fairly interesting image for us, since it has odd dimensions (no power-of-two sizes) and have a great transparent border which use up precious memory meaninglessly. Let's load this sprite into Atlas using the Sprite > Add New Sprite menu item or pressing the + button on the toolbar in the Sprites button group. This will cause as new sprite called star to be created and shown up both in the sprites list and in the sprite sheet editor, will be selected and the settings panel for this sprite will show up in the Inspector:

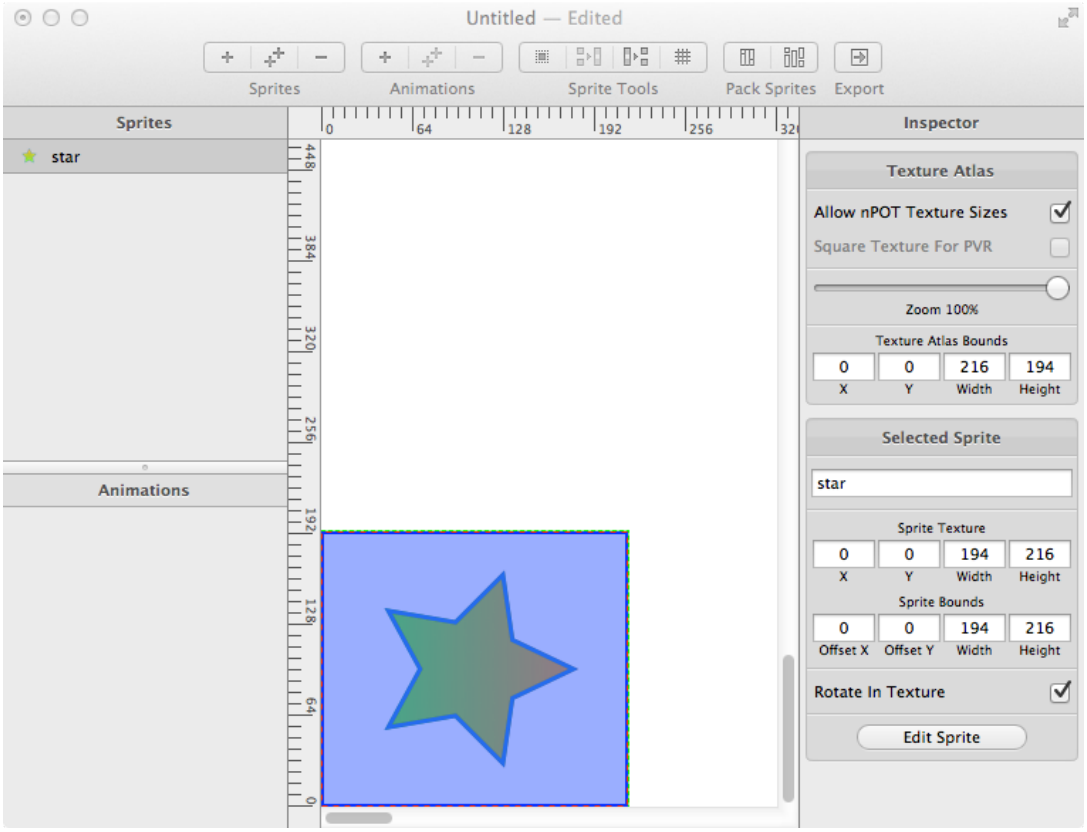


As You can see the selected sprite is highlighted in blue, has a thick blue and a thin green border, and there is a dashed red border around it as well these borders mean the following:

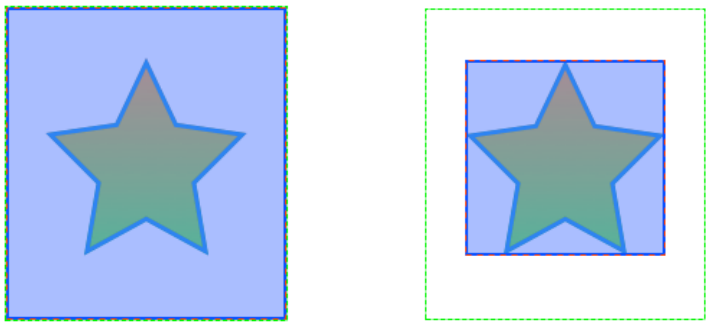
- the thick blue border shows the boundary of the sprite's texture
- the thin dashed green border shows the boundary of the sprite itself
- the dashed red border shows the boundary of the smallest possible sprite sheet according to the parameters set in the Texture Atlas inspector panel

You can see that adding this sprite made the texture atlas's bounds to be 256x256 which is the smallest POT (power-of-two) sized texture which can accommodate our sprite. Now try to set the Allow nPOT Texture Sizes setting in the Texture Atlas inspector panel! You will see that the sprite sheet border now became the same as the sprite boundary.

On the Selected Sprite inspector panel You see the sprite's texture offset and size and the sprite's bounds offset and size (which are now the same) and a checkbox showing Rotate In Texture. Setting this checkbox will rotate the sprite on the sprite sheet only, but will not affect the orientation of the sprite when loaded into a game engine!



As You see rotating a sprite this way will not swap the width and height parameters of the sprite. After rotating the sprite back to it's original form, we will get rid of the unwanted transparent parts of our sprite. Press Sprite -> Trim Transparent Edges or press the trim button on the toolbar's Sprite Tools section:



You can see that the texture part of the sprite shrunk to the smallest possible without losing data but the green sprite bounds stay the same and the remaining texture is positioned so that the sprite will look exactly the same.

3. Editing a sprite

First of all, let's modify the offset of the texture within our sprite! This can be done by three means:

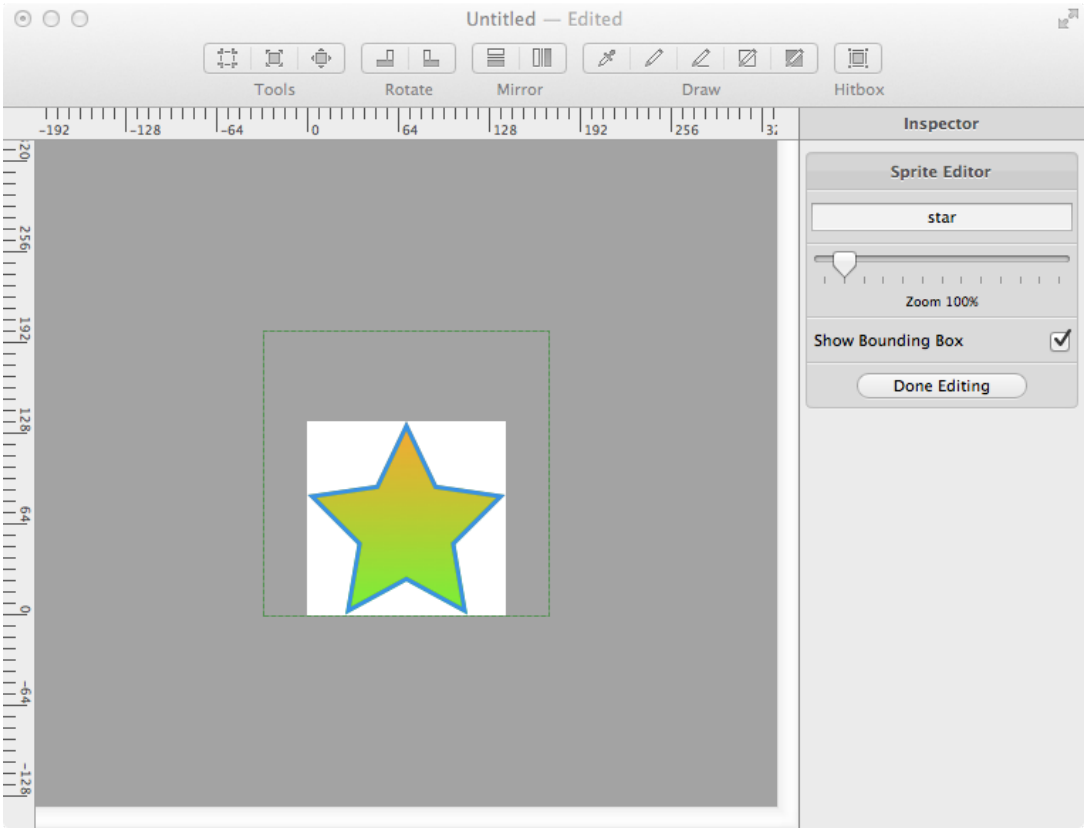
- using the keyboard arrows while holding the COMMAND key will move the sprite within the boundaries. Be aware that the sprite's texture part will not move in the sprite sheet so You will see the green sprite boundary moving.
- entering values into the text fields Sprite Bounds Offset x and Offset y in the Selected Sprite inspector panel and pressing Enter

Also you can modify the sprite bounds size (which cannot be smaller than the sprite texture itself!):

- using the keyboard arrows while holding the COMMAND+ALT key will resize the sprite boundaries.
- entering values into the text fields Sprite Bounds Width and Height in the Selected Sprite inspector panel and pressing Enter

Let's modify these values to offset 30, 0 and size 200, 200.

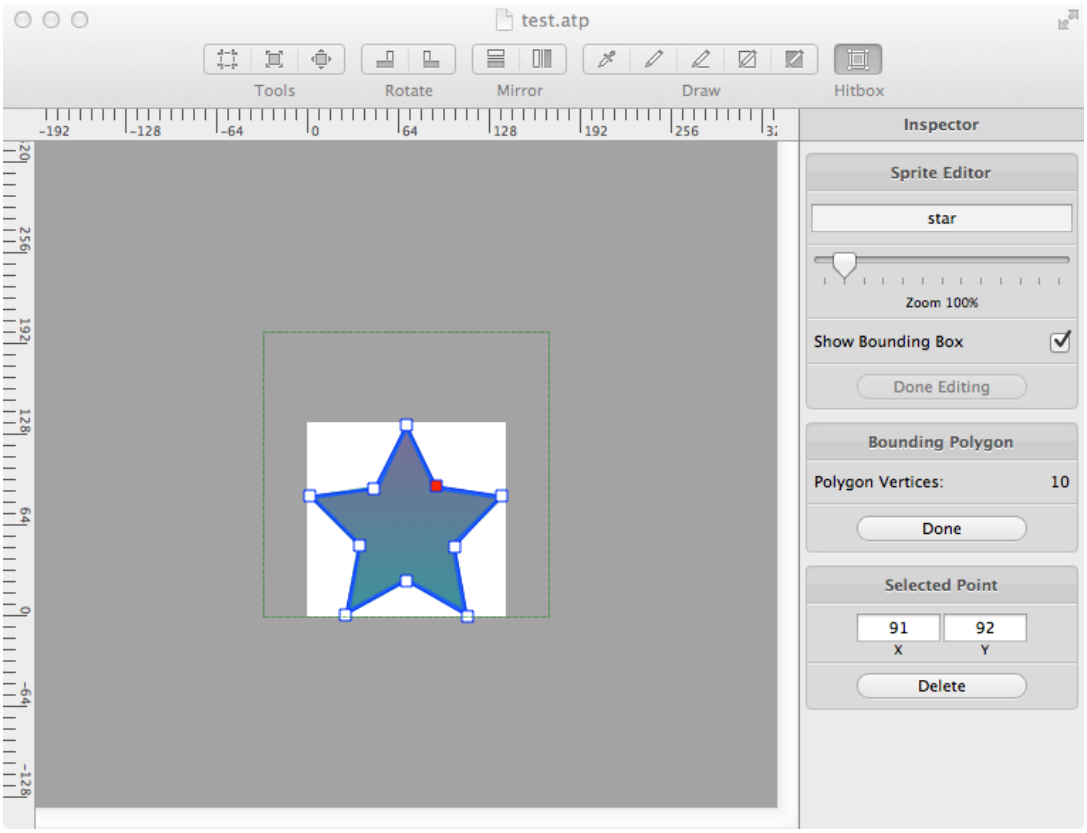
Now open the sprite editor by selecting Sprite > Edit Sprite menu item or pressing the Edit Sprite button on the Selected Sprite inspector panel. Atlas will now switch to sprite editor mode:



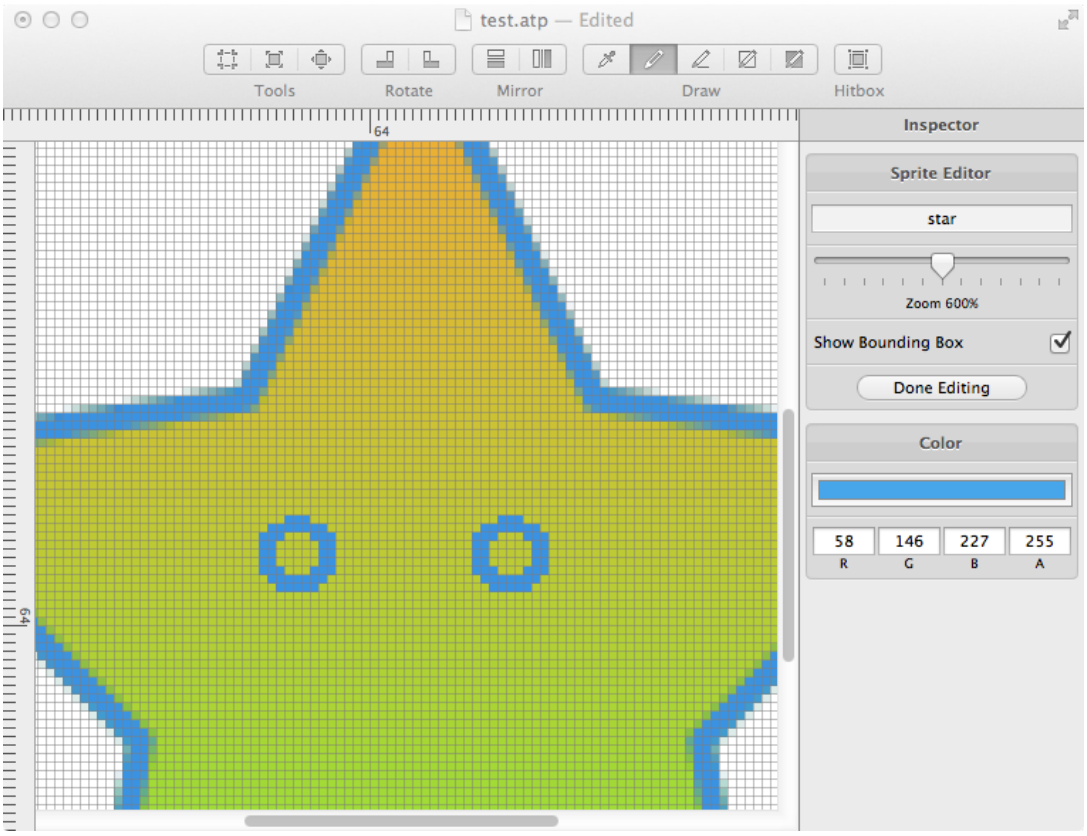
In the Sprite Editor You can alter the sprite's texture the following ways:

- Crop
- Resize texture
- Move texture within its bounds or on the texture canvas
- Rotate
- Mirror
- Change pixel data using basic drawing tools
- Create a hit polygon for the sprite

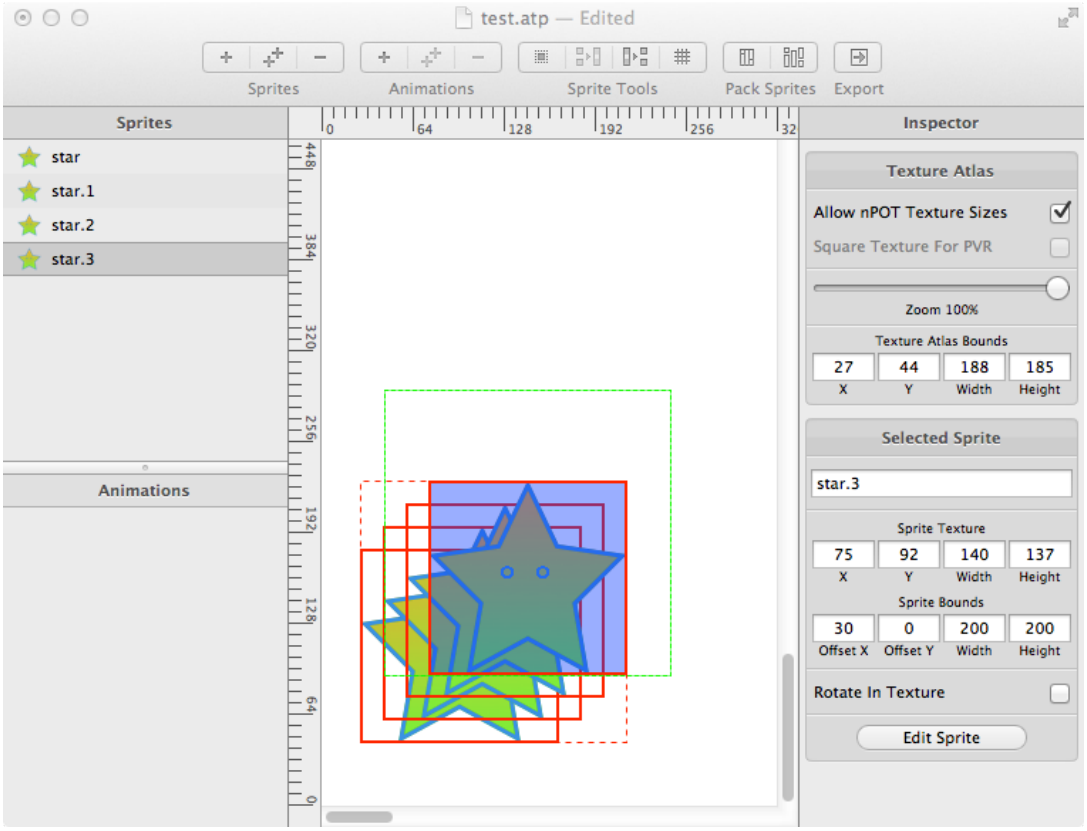
First we will create a hit polygon for our sprite by selecting the Sprite > Edit Sprite Bounds for Hit Test menu item or by pressing the Edit Sprite Bounds toolbar button. You can add new polygon points by clicking on the line between two existing points and move a polygon point by clicking on them and dragging the around with the mouse or moving them with the keyboard arrows and also by writing values into the text fields in the Selected Point inspector panel. Try to make a polygon which follows the star's outline and press the Done button on the Bounding Polygon inspector panel. You can delete the selected point by pressing Delete on the Selected Point inspector panel.



After creating the bounding polygon, draw something, for example two eyes on the sprite texture using the drawing tools. Select the Sprite > Draw Point menu item or press the pencil toolbar icon in the Draw toolbar section, choose a color using the color chooser in the Color inspector panel (or you can use the color picker drawing tool and pick a color from the sprite texture), zoom in to 600% in the Sprite Editor inspector panel to see the individual pixels and draw two circles which will be the eyes of our star sprite. You can also use other drawing tools if you like.



Every change You made in the Sprite Editor can be undone step by step but once You finished editing by pressing the Done Editing button in the Sprite Editor inspector panel, all changes will become one whole undo step in the undo stack of the Sprite Sheet editor.
Now we will create 3 copes of our sprite by selecting the Sprite > Duplicate Selected Sprites menu item or pressing the ++ toolbar button in the Sprites section.

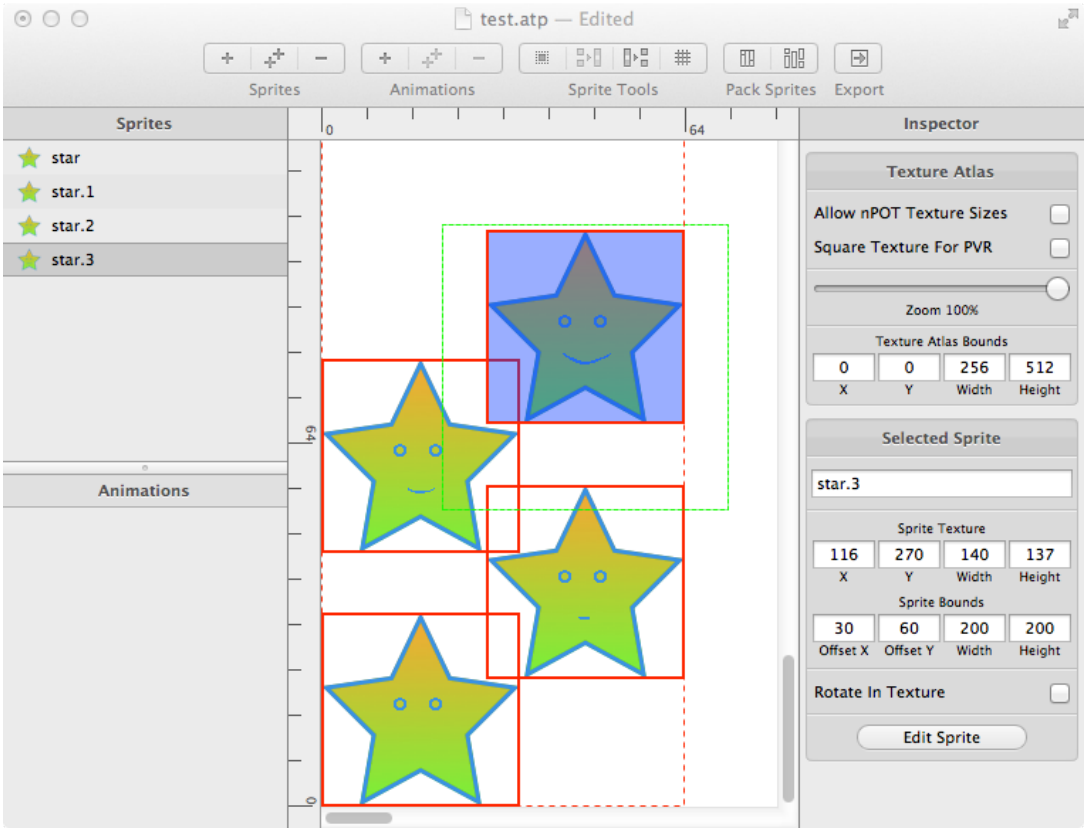


As you can see the 3 newly created sprites will be called star.1, star.2 and star.3. This is because no 2 sprites can be named the same. Also the new sprites overlap with each other. This can be seen by their thick red outline. There are several ways to avoid overlapping:

- move the sprites one by one using the mouse, the arrow keys or writing offset vaues in the Sprite Texture offset fields in the Selected Sprite inspector panel
- pack the sprites using Atlas's pack algorithm, by selecting Sprite > Pack Sprite Loose/ Tight or pressing one of the pack toolbar buttons. Packing tight will pack the sprites so they touch each other, packing loose will let a 2 pixel boundary between sprites.

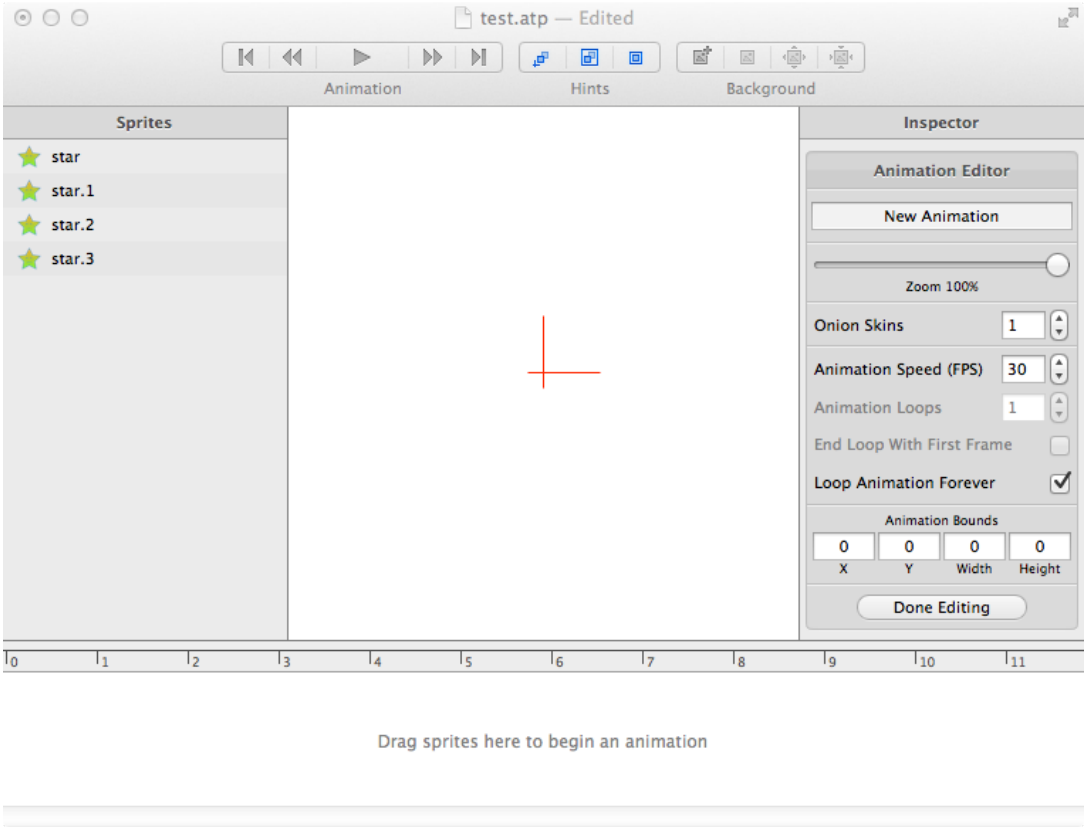
Be aware the fact that overlapping sprites in not always a bad idea. If sprites overlap, the common parts of the overlapping sprites will be seen on both of them if You import the sprite sheet into a game engine, but if the overlapping part in both sprites are transparent it won't affect the output and you might put Your sprites into such positions which will make a smaller sprite sheet. If You pack the 4 start in this demo project and use the POT restriction, the smallest possible sprite sheet is 1024x256, but if You arrange these 4 sprites wisely, You can achieve a 512x256 sprite sheet, which is half the size of the packed one. You can play with sprite rotation as well to achieve even better results.

Now select the newly created sprites one by one and edit them - draw a mouth on each but so that the mouth will be bigger and bigger for each sprite. If You are done, modify the sprite bounds Offset Y for the new sprites to 20, 40 and 60 pixels accordingly. Completing this we will have 4 sprites, each in a different offset and a slightly different texture which will make us a good standing point for our next chapter - making an animation.



4. Creating an animation

Add a new animation by selecting the Animation > Add New Animation menu item or by pressing the + button on the toolbar in the Animations button group. This will cause as new animation called New Animation to be created and shown up in the animations list, will be selected and the settings panel for this animation will show up in the Inspector. Pressing the Edit Animation button in the Selected Animation inspector panel will open the Animation Editor:



Creating an animation is as easy as dragging and dropping frames around. Drag the sprite called star in the Sprite panel to the bottom part of the editor and You will see a nem frame is created. Continue dragging sprites into the sequence and create the following order:

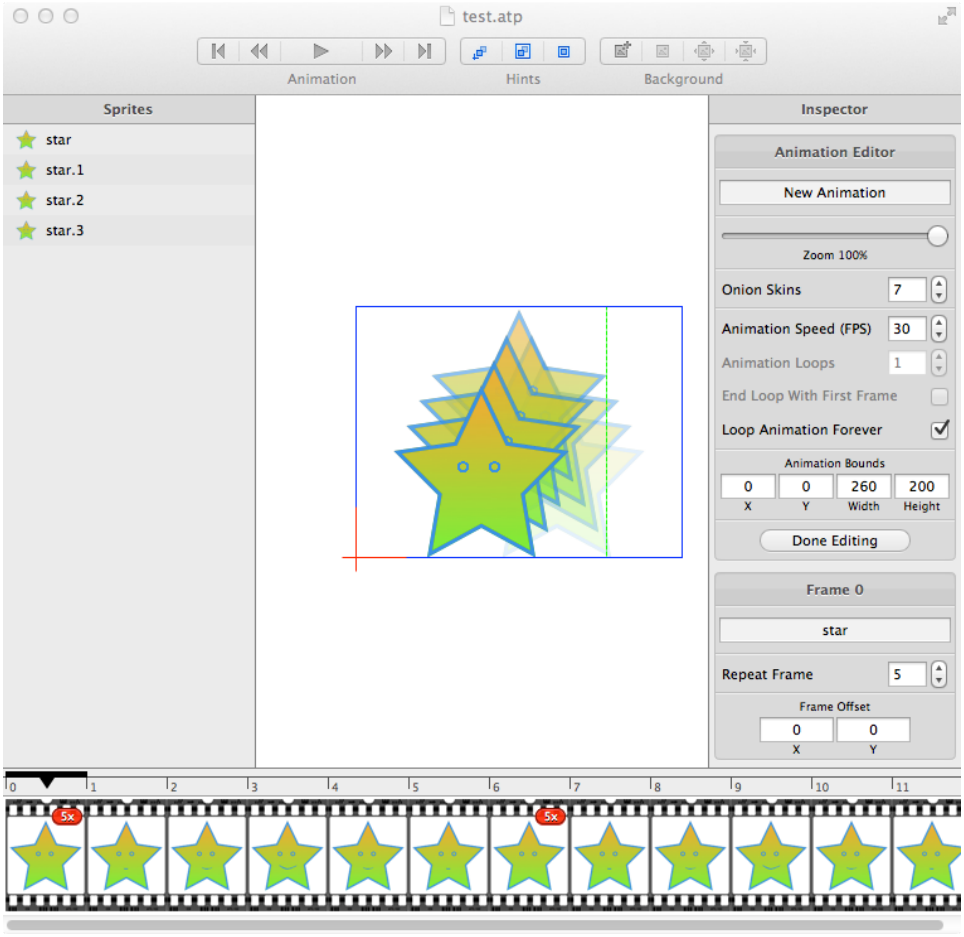
1. star
2. star.1
3. star.2
4. star.3
5. star.2
6. star.1
7. star
8. star.1
9. star.2
10. star.3
11. star.2
12. star.1

This will make an animation which animates our stars two times. You can play with the Animation editor inspector panel's parameters, turn on onion skinning, set the frame rate, loops, etc and play the new animation using the play control on the toolbar or using the keyboard arrows and the space key.

Now set the frame repeat value of frame 0 and frame 6 to 5, and set the Frame Offset X value of the frame to the following:

1. star - 0
2. star.1 - 10
3. star.2 - 20
4. star.3 - 30
5. star.2 - 40
6. star.1 - 50
7. star - 60
8. star.1 - 50
9. star.2 - 40
10. star.3 - 30
11. star.2 - 20
12. star.1 - 10

Now You will have a pretty oscillating animation which uses most of the sprite and animation parameters available.



As You can see, in the animation editor there also are some colored borders around the sprites. These are called hints here and mean the following:

- The red cross means the origin of the animation
- The blue rectangle is the overall bounds of the animation, the sprites move in this rectangle during the animation
- The green rectangle is the bounds of the selected frame

You can switch on and off each of these hints.

You can also import a background image for Your animation which will not be exported in the sprite sheet, it's for design purposes only. You can move and hide this background image as well using the toolbar buttons accordingly.

5. Exporting the spritesheet

When You are finished creating the sprite sheet and the animations, You can export it into various formats using the File > Export Atlas and Spritesheet menu item or pressing the Export toolbar button. You can export the atlas into PNG and TIFF formats, use dithering and reduce the color depth (for TIFF only) to 4 bits/channel, export SD, HD or both, and export to the proprietary atlas plist format or to the standard cocos2d 2.x sprite sheet format.

Be aware that exporting into SD will halve the resolution of the sprite sheet, so for best results use odd sized (sizes dividable by 2) textures and put all your sprites into odd positions (there is a snap tool in the toolbar for that and packing odd sized spritzes will always result in such odd placed sprites).

To move to the next chapter, You must export your work in HD, PNG, non-dithered with cocos2d sprites heet format. This will result in 2 files (assuming you saved Your project as test.atp):

- test_HD.plist - this is the spritesheet metadata
- test_HD.png - this is the spritesheet texture

When you press the export button, a warning message will appear saying you have overlapping sprites, ignore that and press continue as we placed our sprites so the overlapping areas are fully transparent in all our sprites.

6. Creating a cocos2d demo project

Assuming You have knowledge and all the required tools for the cocos2d SDK, follow the steps here to build a minimal “hello world” project using the newly generated sprite sheet.

Create a new project in Xcode using the cocos2d v2.x cocoas2d iOS template.

Add the two files that You exported from Atlas to Your project.

Find the HelloWorldLayer.m source filer in your project.

Add the code found in Appendix A to the source. This code is responsible for loading the animations created with Atlas into the cocos2d environment.

Find the code for init and change to the following:

```
// on "init" you need to initialize your instance
-(id) init
{
	// always call "super" init
	// Apple recommends to re-assign "self" with the "super's" return value
	if( (self=[super init]) ) {

		// load the sprite frames from the test_HD spritesheet
		[[CCSpriteFrameCache sharedSpriteFrameCache]
addSpriteFramesWithFile:@"test_HD.plist"];
		// load the animations from the test_HD spritesheet
		NSDictionary *loopForeverDict = [self
addAtlasAnimationsWithFile:@"test_HD.plist"];
		// create the animation from the one called "New Animaton"
		CCAnimate *anim_action = [CCAnimate actionWithAnimation:[[CCAnimationCache
sharedAnimationCache] animationByName:@"New Animation"]];
		// create an action which repeats it forever
		CCRepeatForever *anim_repeat = [CCRepeatForever actionWithAction:anim_action];
		// create a srite using one of the loaded sprite frames
		CCSprite *s0 = [CCSprite spriteWithSpriteFrame:[[CCSpriteFrameCache
sharedSpriteFrameCache] spriteFrameByName:@"star"]];
		s0.position = ccp( 200, 400 );
		[self addChild:s0];
		// run the animation, forever if the loaded parameters indicate it
		if ((NSNumber*)[[loopForeverDict objectForKey:@"New Animation"]boolValue]) [s0
runAction:anim_repeat];
		else [s0 runAction:anim_action];

	}
	return self;
}
```

Build and run the project. You should see exactly the same animation You created.

The demo projects (both for Atlas and cocos2d) can be downloaded from the Atlas home page: http://ttrgames/atlas

Appendix A: cocos2d source code for Atlas

Add this source code to Your cocos2d project to load Atlas generated animations:

```
// Read an NSDictionary from an Atlas generated cocos2d plist file and parse it
automatically for animations
// Returns a dictionary with keys of the animation names and values of NSNumber (BOOL)
for each animation's loop forever
// parameter (since there is no direct forever looping in cocos2d, loop=0 means don't
animate here unfortunately.)

-(NSDictionary*)addAtlasAnimationsWithFile:(NSString *)plist
{
     NSAssert( plist, @"Invalid texture file name");

    NSString *path = [[CCFileUtils sharedFileUtils] fullPathFromRelativePath:plist];
     NSDictionary *dictionary = [NSDictionary dictionaryWithContentsOfFile:path];

     NSAssert1( dictionary, @"CCAnimationCache: File could not be found: %@", plist);

    NSDictionary *animations = [dictionary objectForKey:@"atlas_animations"];

     if ( animations == nil ) {
              CCLOG(@"cocos2d: CCAnimationCache: No atlas animations were found in
provided dictionary.");
              return nil;
     }

    NSMutableDictionary *loopForeverDict = [[NSMutableDictionary alloc]
initWithCapacity:[animations count]];

    NSArray* animationNames = [animations allKeys];
     CCSpriteFrameCache *frameCache = [CCSpriteFrameCache sharedSpriteFrameCache];

     for( NSString *name in animationNames ) {
              NSDictionary* animationDict = [animations objectForKey:name];
              NSArray* framesArray = [animationDict objectForKey:@"frames"];
              NSNumber *delay = [animationDict objectForKey:@"delay"];
        NSNumber *loopsnr = [animationDict objectForKey:@"loops_nr"];
        NSNumber *loopendsfirstframe = [animationDict
objectForKey:@"loop_ends_first_frame"];
        NSNumber *loopsForever = [animationDict objectForKey:@"loops_forever"];
              CCAnimation* animation = nil;

        [loopForeverDict setObject:loopsForever forKey:name];

              if ( [framesArray count]==0 ) {
                       CCLOG(@"cocos2d: CCAnimationCache: Animation '%@' found in
dictionary without any frames - cannot add to animation cache.", name);
                       continue;
              }

              NSMutableArray *frames = [NSMutableArray arrayWithCapacity:[framesArray
count]];

              for( NSDictionary* frameDict in framesArray) {
            NSString *frameName = [frameDict objectForKey:@"name"];
            NSNumber *repeatFrame = [frameDict objectForKey:@"repeat"];
            CGPoint frameOffset = CCPointFromString([frameDict objectForKey:@"offset"]);
            // unfortunately cocos2d does not support animation frame
              //offsets, so create a copy of the frame in the cache since
            // there may be frames using the same sprite but at
              //different offset
                       CCSpriteFrame *spriteFrame = [[frameCache
spriteFrameByName:frameName] copy];
                       CGPoint newSpriteOffset = [spriteFrame offset];
                  newSpriteOffset.x += frameOffset.x;
                  newSpriteOffset.y += frameOffset.y;
                  [spriteFrame setOffset:newSpriteOffset];
```

```
                    if ( ! spriteFrame ) {
                            CCLOG(@"cocos2d: CCAnimationCache: Animation '%@' refers to
frame '%@' which is not currently in the CCSpriteFrameCache. This frame will not be
added to the animation.", name, frameName);

                            continue;
                    }

                    CCAnimationFrame *animFrame = [[CCAnimationFrame alloc]
initWithSpriteFrame:spriteFrame delayUnits:[repeatFrame unsignedIntegerValue]
userInfo:nil];

            [frames addObject:animFrame];
                    [animFrame release];
             }

             if ( [frames count] == 0 ) {
                    CCLOG(@"cocos2d: CCAnimationCache: None of the frames for animation
'%@' were found in the CCSpriteFrameCache. Animation is not being added to the Animation
Cache.", name);
                    continue;
             } else if ( [frames count] != [framesArray count] ) {
                    CCLOG(@"cocos2d: CCAnimationCache: An animation in your dictionary
refers to a frame which is not in the CCSpriteFrameCache. Some or all of the frames for
the animation '%@' may be missing.", name);
             }

             animation = [CCAnimation animationWithAnimationFrames:frames delayPerUnit:
[delay floatValue] loops:[loopsnr unsignedIntegerValue]];

             [animation setRestoreOriginalFrame:[loopendsfirstframe boolValue]];
             [[CCAnimationCache sharedAnimationCache] addAnimation:animation name:name];
     }

    return loopForeverDict;
}
```