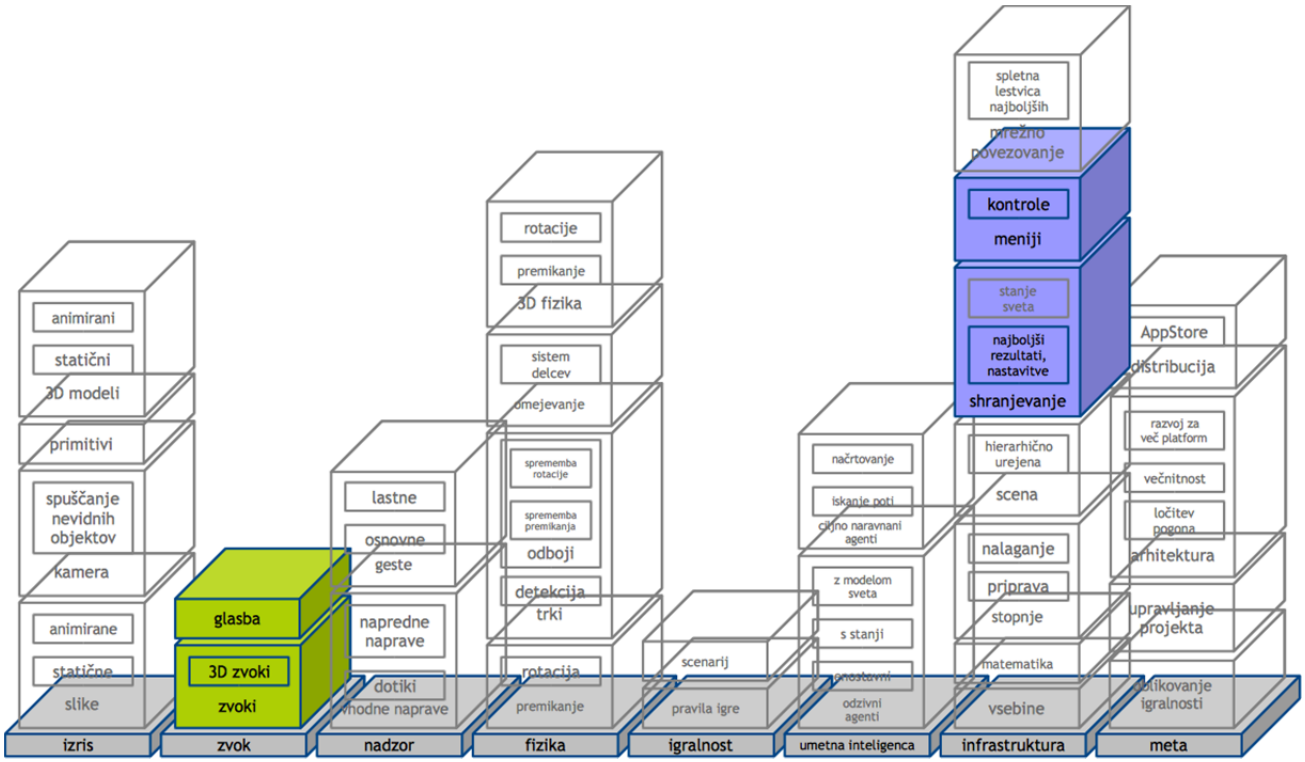
Sklop 6 (nadaljevanje)



Meniji

Pri izdelovanju uporabniških vmesnikov se pokaže, da izdelovanje iger ni samo zabava, temveč je potrebno kdaj tudi stisniti zobe, da se prebijemo do cilja. Čeprav ni tako hudo, kot se morda sliši, programiranje napisov, gumbov, drsnikov in podobnih prvin grafičnega vmesnika ne vsebuje kakšne posebne ustvarjalnosti in zabave, kot je značilno za izdelovanje igralnosti.

Najlažji pristop k arhitekturi vmesnika je, če si ga predstavljamo kot novo "igro" — novo sceno nad obstoječim pogledom na igrin svet. Na tej sceni se namesto entitet igre nahajajo kontrole, ki jih sprogramiramo enako kot entitete. Imajo svojo lokacijo na ekranu in obliko, s katero si pomagamo pri zaznavanju pritiskov. Nastavimo jim lahko hitrost in jih vključimo v fizikalni pogon, da se po potrebi premikajo. V posebnem delu grafičnega pogona poskrbimo za njihov izris.

Izpis besedila

Pri izrisu se pogosto pojavi potreba po izpisu besedila. Igre besedilo izpisujejo tako, da iz teksture, na kateri so vse potrebne črke, izrišejo eno zraven druge, da sestavijo željen napis. V XNI je ta funkcionalnost že vključena v metodi *drawString* razreda *SpriteBatch*. Pred tem pa moramo v igro naložiti posebej pripravljeno teksturo, iz katere bo sistem vedel, kateri deli predstavljajo katero črko.

Teksture s črkami lahko ustvarite z naslednjimi orodji:

- Bitmap font maker
- SpriteFont 2
- Fancy Bitmap Font Generator

V vsakem primeru morate dobiti teksturo, kjer so z roza barvo ločene črke:

Črke so običajno bele ali senčene z odtenki sivine, da jih lahko s parametrom *tint* pobarvate v želeno barvo. Najbolje je, če kanal alpha (primeren format je PNG) določa prosojnost črk. Rožnata barva mora biti pri tem neprosojna (RGBA: 255, 0, 255, 255). Če uporabljate format BMP, bo cevovod vsebine sam izločil povsem črne slikovne točke kot prozorne.

S teksturo dodano v projekt jo morate naložiti na poseben način, saj bi se drugače interpretirala kot navadna tekstura. Želimo torej, da cevovod vsebine uporabi drug način procesiranja, ki bo analiziral postavitev črk in ustvaril objekt tipa SpriteFont:

```
FontTextureProcessor *fontProcessor = [[[FontTextureProcessor alloc] init] autorelease];
SpriteFont *font = [self.game.content load:@"FontTexture" processor:fontProcessor];
```

Možno je nastaviti tudi razmak med vrsticami:

```
font.lineSpacing = 14;
```

Za izris uporabimo ukaz *drawString*:

```
[spriteBatch drawStringWithSpriteFont:font text:@"Hello!" to:position tintWithColor:color];
```

Enostavne kontrole

Zdaj si lahko pripravimo kontrolo Label, ki predstavlja napis na nekem mestu:

```
@interface Label : NSObject {
	SpriteFont *font;
	NSString *text;

	Color *color;

	Vector2 *position;
	Vector2 *origin;
	Vector2 *scale;
	float rotation;
	float layerDepth;

	HorizontalAlign horizontalAlign;
	VerticalAlign verticalAlign;
}

- (id) initWithFont:(SpriteFont*)theFont text:(NSString*)theText position:(Vector2*)thePosition;

@property (nonatomic, retain) SpriteFont *font;
@property (nonatomic, retain) NSString *text;

@property (nonatomic, retain) Color *color;

@property (nonatomic, retain) Vector2 *position;
@property (nonatomic, retain) Vector2 *origin;
@property (nonatomic, retain) Vector2 *scale;

@property (nonatomic) float rotation;
@property (nonatomic) float layerDepth;

@property (nonatomic) HorizontalAlign horizontalAlign;
@property (nonatomic) VerticalAlign verticalAlign;

- (void) setScaleUniform:(float)value;

@end

@interface Label ()

- (void) updateOrigin;

@end

@implementation Label

- (id) initWithFont:(SpriteFont*)theFont text:(NSString*)theText position:(Vector2*)thePosition
{
	self = [super init];
	if (self != nil) {
		font = [theFont retain];
		text = [theText retain];
		position = [thePosition retain];

		color = [[Color white] retain];
		origin = [[Vector2 zero] retain];
		scale = [[Vector2 one] retain];

		[self updateOrigin];
	}
	return self;
}

@synthesize font, text, color, position, origin, scale, rotation, layerDepth, horizontalAlign, verticalAlign;

- (void) setFont:(SpriteFont *)value {
	[value retain];
	[font release];
	font = value;
	[self updateOrigin];
}

- (void) setText:(NSString *)value {
	[value retain];
	[text release];
	text = value;
```

```
	[self updateOrigin];
}

- (void) setOrigin:(Vector2 *)value {
	[value retain];
	[origin release];
	origin = value;
	horizontalAlign = HorizontalAlignCustom;
	verticalAlign = VerticalAlignCustom;
}

- (void) setHorizontalAlign:(HorizontalAlign)value {
	horizontalAlign = value;
	[self updateOrigin];
}

- (void) setVerticalAlign:(VerticalAlign)value {
	verticalAlign = value;
	[self updateOrigin];
}

- (void) setScaleUniform:(float)value {
	scale.x = value;
	scale.y = value;
}

- (void) updateOrigin {
	Vector2 *size = [font measureString:text];

	switch (horizontalAlign) {
		case HorizontalAlignLeft:
			origin.x = 0;
			break;
		case HorizontalAlignCenter:
			origin.x = size.x / 2;
			break;
		case HorizontalAlignRight:
			origin.x = size.x;
			break;
	}

	switch (verticalAlign) {
		case VerticalAlignTop:
			origin.y = 0;
			break;
		case VerticalAlignMiddle:
			origin.y = size.y / 2;
			break;
		case VerticalAlignBottom:
			origin.y = size.y;
			break;
	}
}

- (void) dealloc
{
	[font release];
	[text release];
	[color release];
	[position release];
	[origin release];
	[scale release];

	[super dealloc];
}

@end
```

Na podoben način sestavimo še druge kontrole, recimo sliko (primerno za prikaz slikovnih prvin grafičnega vmesnika). Oboje potem iz novega razreda za izris (GuiRenderer) izrišemo:

```
@interface GuiRenderer : DrawableGameComponent {
	SpriteBatch *spriteBatch;
```

```
	id<IScene> scene;
}

- (id) initWithGame:(Game*)theGame scene:(id<IScene>)theScene;

@end

@implementation GuiRenderer

- (id) initWithGame:(Game*)theGame scene:(id<IScene>)theScene
{
	self = [super initWithGame:theGame];
	if (self != nil) {
		scene = theScene;
	}
	return self;
}

- (void) loadContent {
	spriteBatch = [[SpriteBatch alloc] initWithGraphicsDevice:self.graphicsDevice];
}

- (void) drawWithGameTime:(GameTime *)gameTime {
	[spriteBatch beginWithSortMode:SpriteSortModeDeffered
				BlendState:nil
			SamplerState:[SamplerState pointClamp]
		DepthStencilState:nil
		   RasterizerState:nil];

	for (id item in scene) {
		Label *label = [item isKindOfClass:[Label class]] ? item : nil;
		Image *image = [item isKindOfClass:[Image class]] ? item : nil;

		if (label) {
			[spriteBatch drawStringWithSpriteFont:label.font text:label.text
								to:label.position tintWithColor:label.color
						rotation:label.rotation origin:label.origin
							scale:label.scale effects:SpriteEffectsNone
						layerDepth:label.layerDepth];
		}

		if (image) {
			[spriteBatch draw:image.texture to:image.position
			     fromRectangle:image.sourceRectangle tintWithColor:image.color
					rotation:image.rotation origin:image.origin
					    scale:image.scale effects:SpriteEffectsNone
				    layerDepth:image.layerDepth];
		}
	}

	[spriteBatch end];
}

- (void) unloadContent {
	[spriteBatch release];
}

@end
```

Sestavljene kontrole

Kompleksnejše kontrole lahko običajno sestavimo s kombinacijo preprostejših. Če uporabimo sliko gumba in čez njega postavimo napis, lahko izrišemo gumb s poljubnim napisom. Polje za obkljukanje (checkbox) lahko nato sestavimo iz gumba z dvema slikama (kvadratek za ozadje in kljukica) ter besedilom poleg.

Pri tem moramo kontrole vključiti v procesiranje, saj morajo pregledovati, če je uporabnik izvedel pritisk na njihovi lokaciji. Gumb z osnovno funkcionalnostjo bi izgledal takole:

```
@interface Button : NSObject <ISceneUser> {
	id<IScene> scene;
```

```
	Image *backgroundImage;
	Label *label;

	Rectangle *inputArea;
	BOOL enabled;

	BOOL isDown;
	BOOL wasPressed;
	BOOL wasReleased;
	int pressedID;

	Color *labelColor, *labelHoverColor, *backgroundColor, *backgroundHoverColor;
}

- (id) initWithInputArea:(Rectangle*)theInputArea background:(Texture2D*)background font:(SpriteFont *)font
text:(NSString *)text;

@property (nonatomic, readonly) Rectangle *inputArea;
@property (nonatomic) BOOL enabled;

@property (nonatomic, readonly) BOOL isDown;
@property (nonatomic, readonly) BOOL wasPressed;
@property (nonatomic, readonly) BOOL wasReleased;

@property (nonatomic, readonly) Image *backgroundImage;
@property (nonatomic, readonly) Label *label;

@property (nonatomic, retain) Color *labelColor, *labelHoverColor, *backgroundColor, *backgroundHoverColor;

- (void) update;

@end

@implementation Button

- (id) initWithInputArea:(Rectangle*)theInputArea
	background:(Texture2D*)background
		font:(SpriteFont*)font
		text:(NSString*)text
{
	self = [super init];
	if (self != nil) {

		inputArea = [theInputArea retain];
		enabled = YES;

		backgroundImage = [[Image alloc] initWithTexture:background position:[Vector2
vectorWithX:inputArea.x y:inputArea.y]];
		label = [[Label alloc] initWithFont:font
							text:text
					position:[Vector2 vectorWithX:inputArea.x + 10 y:inputArea.y +
inputArea.height/2]];
		label.verticalAlign = VerticalAlignMiddle;

		self.backgroundColor = [Color white];
		self.backgroundHoverColor = [Color dimGray];

		self.labelColor = [Color black];
		self.labelHoverColor = [Color white];

	}
	return self;
}

@synthesize inputArea, enabled, isDown, wasPressed, wasReleased, scene, backgroundImage, label;
@synthesize labelColor, labelHoverColor, backgroundColor, backgroundHoverColor;

- (void) setLabelColor:(Color *)value {
	[value retain];
	[labelColor release];
	labelColor = value;
	label.color = labelColor;
}

- (void) setBackgroundColor:(Color *)value {
```

```
	[value retain];
	[backgroundColor release];
	backgroundColor = value;
	backgroundImage.color = backgroundColor;
}

- (void) addedToScene:(id <IScene>)theScene {
	// Add child items to scene.
	[theScene addItem:backgroundImage];
	[theScene addItem:label];
}

- (void) removedFromScene:(id <IScene>)theScene {
	// Remove child items.
	[theScene removeItem:backgroundImage];
	[theScene removeItem:label];
}

- (void) update {
	if (!enabled) {
		return;
	}

	TouchCollection *touches = [TouchPanelHelper getState];
	if (!touches) {
		return;
	}

	BOOL wasDown = isDown;

	isDown = NO;
	wasPressed = NO;
	wasReleased = NO;

	for (TouchLocation *touch in touches) {
		if ([inputArea containsVector:touch.position] && touch.state != TouchLocationStateInvalid) {
			if (touch.state == TouchLocationStatePressed) {
				pressedID = touch.identifier;
				wasPressed = YES;
			}

			// Only act to the touch that started the push.
			if (touch.identifier == pressedID) {
				if (touch.state == TouchLocationStateReleased) {
					wasReleased = YES;
				} else {
					isDown = YES;
				}
			}
		}
	}

	if (isDown && !wasDown) {
		backgroundImage.color = backgroundHoverColor;
		label.color = labelHoverColor;
	} else if (!isDown && wasDown) {
		backgroundImage.color = backgroundColor;
		label.color = labelColor;
	}
}

- (void) dealloc
{
	[labelColor release];
	[labelHoverColor release];
	[backgroundColor release];
	[backgroundHoverColor release];
	[backgroundImage release];
	[label release];
	[inputArea release];
	[super dealloc];
}

@end
```

Gumb uporablja protokol *ISceneUser*, tako da izve, kdaj smo ga dodali na sceno GUI-ja. Takrat doda svoji enostavni kontroli na sceno, ki jih bo nato avtomatično izrisal *GuiRenderer*, ki se sprehaja čez objekte na sceni.

Ko na gumbu pokličemo *update*, bo ta posodobil stanje spremenljivk *isDown*, *wasPressed* in *wasReleased*, na katere lahko potem primerno reagiramo (gremo v nov meni, začnemo igro ...), recimo:

```
Button *back;

- (void) updateWithGameTime:(GameTime *)gameTime {
	// Update all buttons.
	for (id item in scene) {
		Button *button = [item isKindOfClass:[Button class]] ? item : nil;

		if (button) {
			[button update];
		}
	}

	if (back.wasReleased) {
		// Go back to main menu.
	}
}
```

Naloga: kontrole

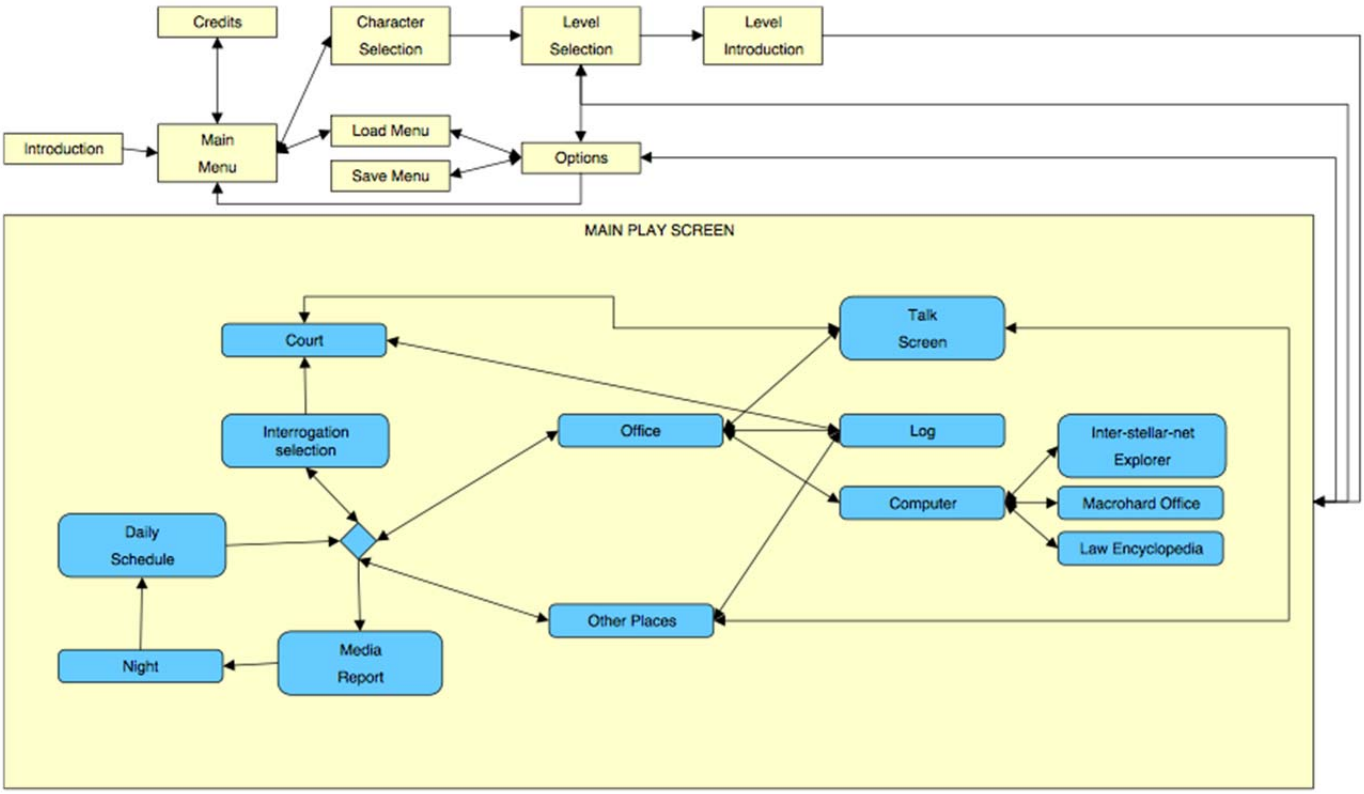
Podobno kot svet igre sestavlja scena z različnimi objekti, tako je uporabniški vmesnik sestavljen iz različnih kontrol. To so dobro poznani gradniki: napis, slika, gumb, vnosno polje, izbira med več možnostmi, preklapljanje med dvema možnostima, izbira številske vrednosti itd.

Na najnižjem nivoju potrebujemo običajno zgolj napis in sliko, ki vsebujeta vse potrebne lastnosti za izris. Ostale kontrole so nato sestavljene iz več napisov in slik ter kontrolne logike, ki po potrebi spreminja svoje sestavne dele.

Na začetku kontrole najlažje izdelamo za prikaz vmesnika med samim igranjem (recimo prikaz točk, izris slik za življenja, gumb za resetiranje stopnje). V razred Gameplay dodamo sceno za vmesnik in nov GuiRenderer, ki zna izrisati kontrole. Naloga je opravljena, ko dodate vsaj eno vrsto kontrole.

Stanja igre

Igra se seveda ne odvija zgolj na igralnem zaslonu. S kontrolami smo za začetek sestavili vmesnik na njemu, a v glavnem jih bomo uporabili za sestavljanje menijev, ki se prikažejo pred igranjem in po igranju. Različnim menijem in ločenim delom igralnosti (izbira stopnje, glavni igralni zaslon, nakupovanje nadgradenj, poročilo o uspešnosti opravljanja stopnje ...) rečemo stanja igre. Običajno je igra samo v enem stanju, med njimi pa prehaja po različnih povezavah.



Za tako kompleksen graf prehajanja med stanji moramo narediti pravi avtomat, kjer vsako stanje sporoči, v katerega naj se nadaljuje.

Za enostavnejše primere, kjer je graf v obliki drevesa (glavni meni iz katerega se spuščamo vedno bolj globoko v podmenije in nato vračamo nazaj) nam prav pride že enostaven sklad.

V vsakem primeru bo naš trenutni razred Gameplay postal le eno od stanj igre, kateremu dodamo ostala stanja (MainMenu, Options ...). Pripravimo si abstraktni razred *GameState*, ki bo vseboval metodi *activate* in *deactive*:

```
@interface GameState : GameComponent {
	OurGameClass *ourGame;
}

- (void) activate;
- (void) deactivate;

@end

@implementation GameState

- (id) initWithGame:(Game *)theGame
{
	self = [super initWithGame:theGame];
	if (self != nil) {
		ourGame = (OurGameClass*)self.game;
	}
	return self;
}

- (void) activate {}
- (void) deactivate {}

@end
```

Kot vidite smo si pripravili še spremenljivko ourGame, s katero podobno kot self.game dostopamo do igre, ta je tokrat tudi v pravem tipu, da nam ni potrebno vedno pretvarjati tipa z *(OurGameClass*)*. To nam bo prišlo prav zato, ker bomo v našem glavnem razredu igre spisali zmožnost za prehajanje med stanji, nakar bomo iz konkretnih stanj preko spremenljivke *ourGame* te ukaze klicali.

Začnemo z izdelavo seznama, ki nam bo služil za sklad stanj, ki so na njem:

```
@interface OurGameClass : Game {
	GraphicsDeviceManager *graphics;

	// Game state
	NSMutableArray *stateStack;
}

- (void) pushState:(GameState*)gameState;
- (void) popState;

@end
```

Metodi *pushState* in *popState* bosta s seznama dodajali in odstranjevali stanja. Pri tem bo igra poskrbela, da je aktivno zgolj zgornje stanje:

```
- (void) pushState:(GameState *)gameState {
	GameState *currentActiveState = [stateStack lastObject];
	[currentActiveState deactivate];
	[self.components removeComponent:currentActiveState];

	[stateStack addObject:gameState];
	[self.components addComponent:gameState];
	[gameState activate];
}

- (void) popState {
	GameState *currentActiveState = [stateStack lastObject];
	[currentActiveState deactivate];
	[self.components removeComponent:currentActiveState];
	[stateStack removeLastObject];

	currentActiveState = [stateStack lastObject];
	[self.components addComponent:currentActiveState];
	[currentActiveState activate];
}
```

Preostane nam le še, da v *initialize* v glavnem razredu igre dodamo začetno stanje:

```
- (void) initialize {

	// Start in main menu.
	MainMenu *mainMenu = [[[MainMenu alloc] initWithGame:self] autorelease];
	[self pushState:mainMenu];

	// Initialize all components.
	[super initialize];
}
```

Metoda *pushState* bo stanje potisnilo na sklad in ga dodalo med komponente igre, tako da bo vključeno v igrino zanko. Ob klicu metode activate pa bo stanje poskrbelo, da bo dodatno dodalo med komponente še vse dele, ki se morajo prav tako vključiti v zanko, da stanje pravilno deluje. Recimo za zgornji primer MainMenu:

```
- (void) activate {
	[self.game.components addComponent:scene];
	[self.game.components addComponent:renderer];
}
```

Obratno se seveda zgodi v deactivate. Ko hočemo iti v nov meni, recimo ob pritisku na gumb, kot smo demonstrirali v prejšnjem delu o kontrolah, enostavno dodamo novo stanje (ali pa ga odvzamemo ob vračanju):

```
Button *startGame;
Button *back;

- (void) updateWithGameTime:(GameTime *)gameTime {
	// Update all buttons.
	for (id item in scene) {
		Button *button = [item isKindOfClass:[Button class]] ? item : nil;
```

```
                if (button) {
                        [button update];
                }
        }

        if (startGame.wasReleased) {
                Gameplay *gameplay = [[[Gameplay alloc] initWithGame:self.game] autorelease];
                [ourGame pushState:gameplay];

        }

        if (back.wasReleased) {
                [ourGame popState];
        }
}
```

Naloga: meniji

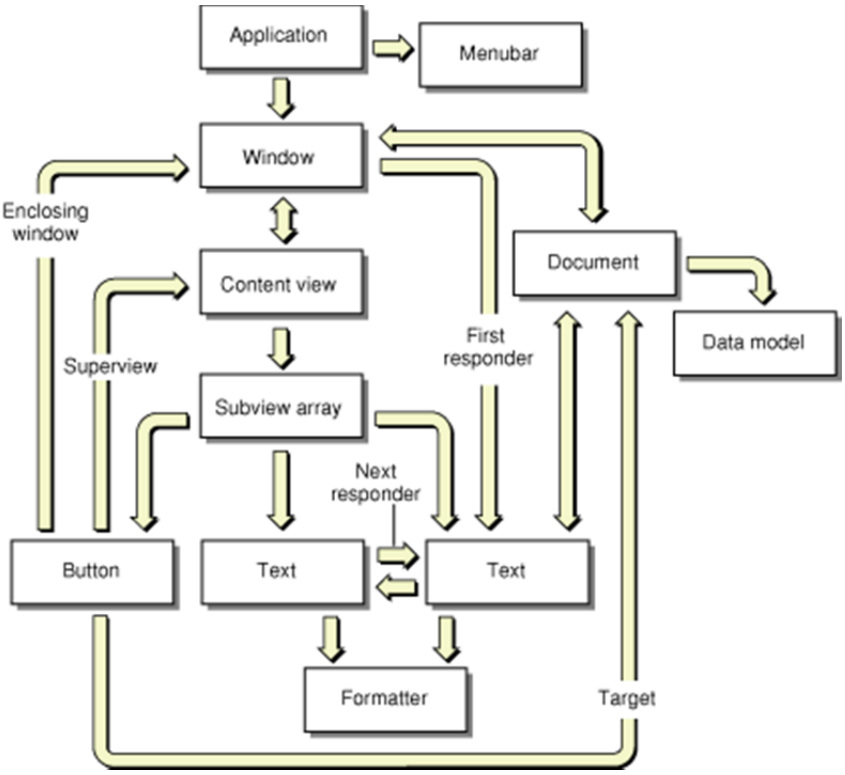
Z izdelanimi kontrolami lahko izdelamo menije, ki sestavljajo stanja naše igre. Določiti moramo prehode med njimi in se glede na kompleksnost povezav odločiti za način, kako bomo skrbeli za menjavanje med stanji.

Najenostavnejša različica je sklad stanj, na katerega nalagamo nova stanja in se nato vračamo na prejšnja. Aktivno je samo zgornje stanje, nekativna pa vmes odstranimo s seznama komponent, tako da se ne izvajajo.

To je seveda le en izmed veliko pristopov k gradnji uporabniških vmesnikov. Sami lahko izdelate menije na kakršenkoli način. Naloga je opravljena, ko imamo pred igranjem vsaj en meni in iz njega pravilno deluje prehod v igranje ter nazaj.

Shranjevanje

Cocoa omogoča shranjevanje oziroma arhiviranje celotnih grafov objektov. Graf objektov je zbirka objektov, ki med sabo kažejo en na drugega.



Graf objektov je tudi naša scena, ki v svojem seznamu hrani povezave do vseh objektov v svetu igre. Čisto na koncu ima lahko graf tudi samo en objekt. Vsak tak graf lahko s pomočjo razreda *NSKeyedArchiver* shranimo in kasneje ponovno naložimo z *NSKeyedUnarchiver*, kar je zelo uporabno na več nivojih v igrah. Če vzamemo stanje celotne scene lahko izdelamo shranjevanje položaja v igri. Enostavnejša je izdelava lestvice najboljših rezultatov, ki jih hranimo v seznamu ali slovarju.

Shranjevanje obstoječih razredov

V vsakem primeru NSKeyedArchiver pričakuje, da mu podamo za arhiviranje glavni objekt, iz katerega se bo nato shranjevanje nadaljevalo na vse vsebovane objekte. Recimo, da hočemo shraniti seznam rezultatov:

```
NSMutableDictionary *scores = [[[NSMutableDictionary alloc] init] autorelease];

[scores setObject:[NSNumber numberWithInt:1000] forKey:@"Alice"];
[scores setObject:[NSNumber numberWithInt:590] forKey:@"Bob"];
[scores setObject:[NSNumber numberWithInt:334] forKey:@"Clint"];
```

Zdaj lahko *scores* enostavno arhiviramo v datoteko, ki jo pred tem pripravimo v uporabniški domeni:

```
NSString *rootPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES)
objectAtIndex:0];
NSString *scoreSaveFile = [rootPath stringByAppendingPathComponent:@"scores"];

[NSKeyedArchiver archiveRootObject:scores toFile:scoreSaveFile];
```

Ker smo v slovar shranili zgolj enostavne objekte, ki jih je Apple že pripravil za shranjevanje, vse skupaj deluje. Slovar lahko enostavno naložimo z ukazom:

```
NSMutableDictionary *loadedScores = [NSKeyedUnarchiver unarchiveObjectWithFile:scoreSaveFile];
```

Shranjevanje lastnih razredov

Če bi hoteli shraniti enega izmed svojih razredov, stvar ni tako samodejna. Kar namreč naredi NSKeyedArchiver, ko poskusi shraniti določen objekt je, da na njem pokliče metodo encode iz protokola NSCoder. V vgrajenih razredih je ta metoda že pripravljena, pri naših pa moramo to narediti ročno. Če bomo graf objektov vključeval katerikoli razred brez vmesnika NSCoder, bo program javil napako. Poskrbeti moramo torej, da se znajo naši razredi shraniti.

Recimo, da smo pripravili razred *GameProgress* s podatki o odklenjenih stopnjah in nasprotnikih:

```
@interface GameProgress : NSObject <NSCoding> {
	int currentLevel;
	NSTimeInterval timePlaying;
	BOOL opponentUnlocked[OpponentTypes];
	Player *playerInfo;
}
```

Na razredu uporabimo protokol NSCoding, za katerega moramo spisati metodi *encodeWithCoder* (shranjevanje) in *initWithCoder* (nalaganje). Pomagamo si z objektom tipa NSCoder, ki je v bistvu slovar tekstovnih nizov v poljubne primitivne tipe ali objekte:

```
- (void) encodeWithCoder:(NSCoder *)aCoder {
	[aCoder encodeInt:currentLevel forKey:@"currentLevel"];
	[aCoder encodeDouble:timePlaying forKey:@"timePlaying"];
	for (int i = 0; i < OpponentTypes; i++) {
		[aCoder encodeBool:opponentUnlocked[i] forKey:[NSString stringWithFormat:@"opponentUnlocked%i",
i]];
	}
	[aCoder encodeObject:playerInfo forKey:@"playerInfo"];
}
```

Obratno storimo pri nalaganju:

```
- (id) initWithCoder:(NSCoder *)aDecoder {
	self = [super init];
	if (self != nil) {
		currentLevel = [aDecoder decodeIntForKey:@"currentLevel"];
		timePlaying = [aDecoder decodeDoubleForKey:@"timePlaying"];
		for (int i = 0; i < OpponentTypes; i++) {
			opponentUnlocked[i] = [aDecoder decodeBoolForKey:[NSString
stringWithFormat:@"opponentUnlocked%i", i]];
		}
		playerInfo = [aDecoder decodeObjectForKey:@"playerInfo"];
	}
	return self;
}
```

Na tak način lahko shranimo in naložimo vsakršen objekt, razen glavnega razreda igre in komponent, ki protokola NSCoding ne implementirajo. Pozorni moramo biti torej, da v našem grafu objektov ne poskušamo shraniti nobene komponente in povezave do igre.

Naloga: najboljši rezultati, nastavitve

Najenostavnejši del shranjevanja je serializacija že obstoječih podatkovnih tipov, kot so *NSMutableDictionary* ali *NSMutableArray*. Tako si lahko najboljše rezultate shranimo v podatkovno strukturo slovarja in za vsako ime (*NSString*) shranimo dosežen rezultat (*NSNumber*). Podobno lahko pod vnaprej definirane ključe (besede) shranimo vrednosti nastavitev, recimo, ali imamo vklopljen zvok ali ne (pod ključ *@"SoundOn*" shranimo *BOOL* vrednost v objektu *NSNumber*).

Ko imamo tako pripravljeno podatkovno strukturo uporabimo *NSKeyedArchiver* in *NSKeyedUnarchiver*, da v datoteko (pot dobimo s pomočjo *NSSearchPathForDirectoriesInDomains*) shranimo in na drugi strani iz nje naložimo podatke. Tako dosežemo, da se rezultati ali nastavitve ohranijo tudi po tem, ko zapremo aplikacijo.