Authors:	Robert Jäger
Archive:	
Version:	0.1
Status	Draft

Abstract: This document specifies the functionality of the Dark Tower game, which is part of the Games component

Keywords: **Games, Dark Tower, Entertainment**

Distribution List:

Table 1: Revision History

Version	Date	Change description

Changes from the previous version are indicated with change bars to the left of the text.

Table 2: Review History

Version	Date	Review result
1.0		Initial Release

The following persons have read and are committed to this:

Table 3: Acceptance

Name	Date	Signature

Table of Contents

List of tables

List of figures

1 Introduction

1.1Scope

This document describes the internal structure of the Dark Tower game and all algorithms used in detail as well as all components required by the game. The rules of the game are included as well.

1.2Purpose

This specification is intended to make the structure of game clear and understandable for developers who are extending, changing or reusing parts of the game source.

1.3Overview of document

Chapter 2 gives the applicable documents
Chapter 3 gives a general description of the subject
Chapter 4 describes the requirements
Chapter 5 reveals the internal structure
Chapter 6 describes the game
Chapter 7 informs about remarks

1.4Acronyms, Abbreviations and Terminology

BMP2ICON	Tool for converting 24Bit Windows Bitmaps to the Ghandler icon format
BMP2MAP	Tool for converting 24Bit Windows Bitmaps to the game level format
CSI	Customer Software Interface
DOS	Disk Operating System
GSW	General Software
MS	Microsoft
OPA	Open Peripheral Architecture
OSD	on Screen Display
RGB	Red Green Blue
Target	The hardware the game is running on
UNIX	Well known operating system
Vectrex	Old Entertainment System

2 Applicable Documents

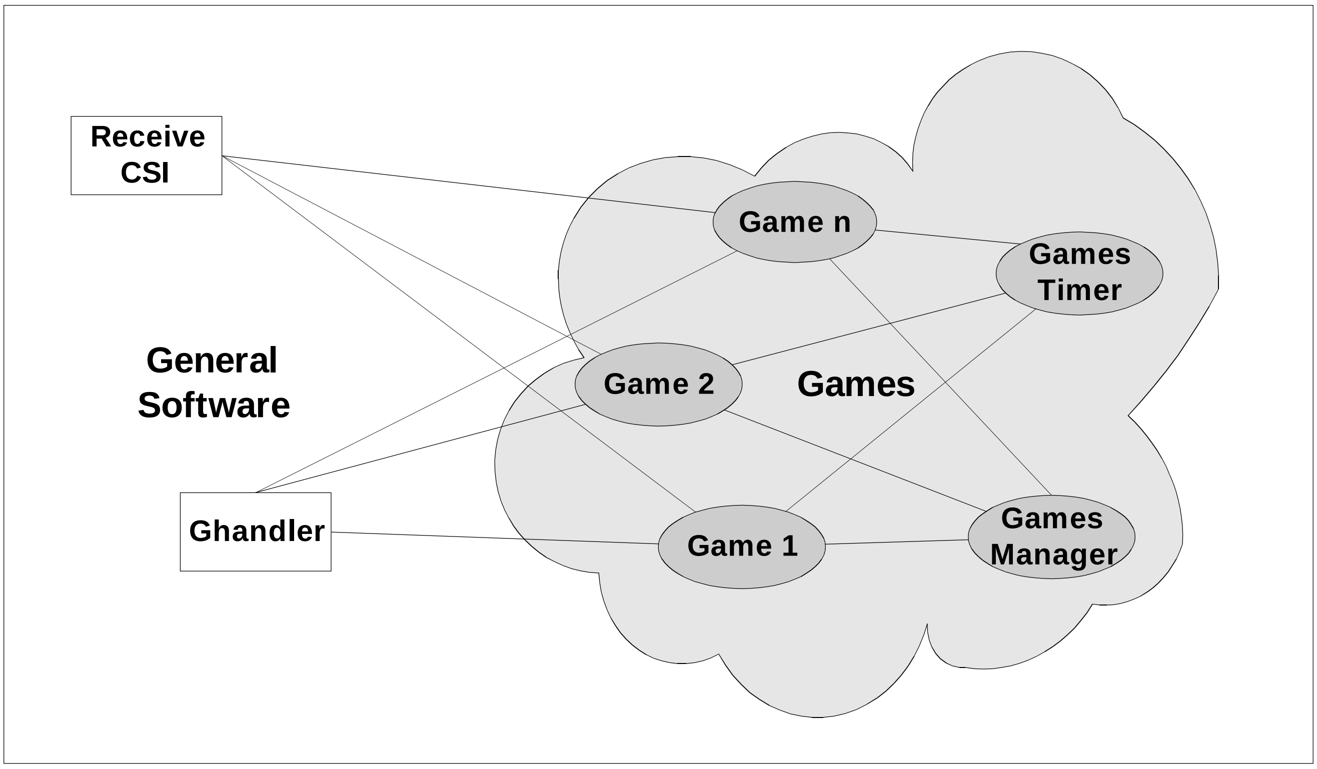
2.1References

[R-1] Graphic library, G. Käser, 1992
[R-2] Computergrafik, mitp, 2003
[R-3] Computer Graphics – Principles and Practice, Addison Wesley, 1990

3 Overall Description

3.1Product perspective

Figure 1: Context diagram



The Game is activated by the Games Manager.
The Games Timer is part of the Games Manager and is used for getting a time dependent factor. This allows time correction for all movement routines in the game.
The remote signals are retrieved from receive CSI and then handled within the game while the graphic data is prepared internally and then forwarded to the Ghandler for graphical output.

4 Requirements

4.1Game content requirements

The game needs to be easy to understand with a low learning curve. The typical player does not want to read a manual to have a quick game; it must be playable right away.
A suitable game needs a high motivation curve, considering the fact that playing it is not very complicated.
It is required that the game can be quit at any time, because playing games while driving is not allowed.
There should be some random factors in the game, which motivate the user to play the game again and again.
Also the game controls need to be simple, the whole input is done with the target remote, which only has a limited number of buttons.

4.2Code requirements

Since the target device is very limited in memory, the game should be as small as possible. The final code size is limited by the available development time as well.
Structures and routines should be reusable by future games and realized as shared components as far as possible. The code should be structured clearly and commented for future improvements, changes and reuse of code parts.
The whole game has to run time corrected, so that the speed of any motion will stay the same on future hardware releases.

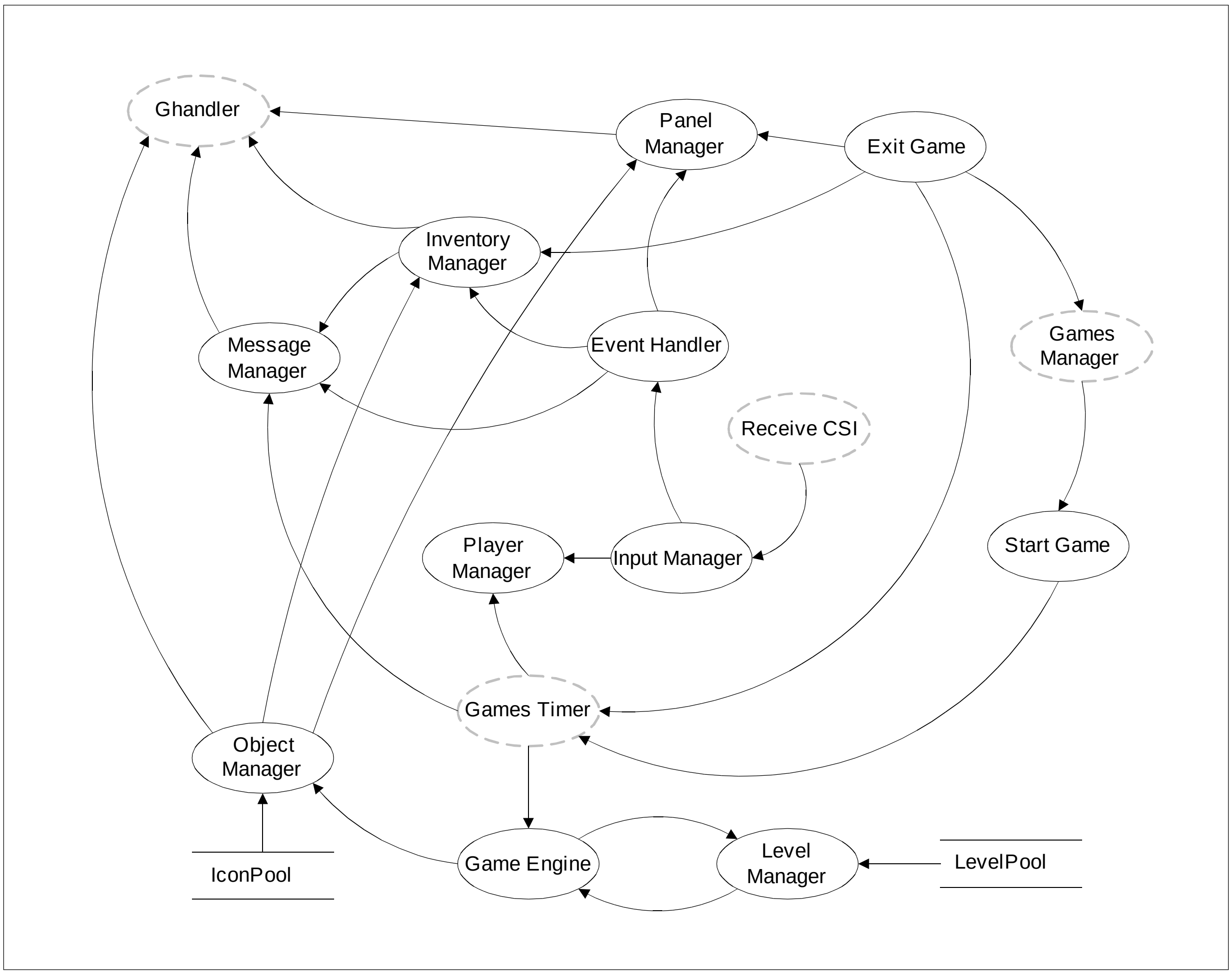
4.3Suitable game

Dark Tower is a game that matches these requirements. The original Vectrex version is based on a cardboard game and was never released officially.
The graphics are based on vector shapes and can be displayed easily with the graphic library of the GSW.
The game content can be changed or extended; there is a lot of room for new features, which makes possible copyright problems less relevant. This is also very helpful for finishing the project in time. Depending on the time left, features can be added or left out, without making the game senseless.
Dark Tower provides a simple, but catching game content. The replay value is very high.

5 Game structure

5.1Code structure

Figure 2: Code structure



The game divided into several managers and handlers with specific tasks. This makes the code better understandable, because only routines that are related to each other will be found in the same manager.
In the following section, all parts of the game are described in detail.

5.1.1Game Start

Before the game can be played, several things have to be initialized.
The games_timer needs to be started and run for constant frame rate, all panel objects have to be created, the ghandler needs to be initialized, and several variables need to be set.
After all initialization processes are finished, the game loop is called. The game loop takes care that all managers and functions are called and executed in the correct order. The inventory and panels have to be drawn over the 3D view, text should not appear behind trees, and the latest timer value has to be read from the games_timer. This will happen as long as the game is running. The game is ready for playing.

5.1.2Game Exit

When the game was exited, the memory has to be cleared and the games_timer has to be stopped.
Freeing the memory occupied by the Panel Manager and the Inventory Manager is done very quickly, because all objects are arranged in linked lists.
After stopping the games_timer, the Games Manager has to be activated again. Afterwards the player will find himself back in the secret games menu.

5.1.3Input Manager

The Input Manager receives remote signals from receive CSI and reacts on those. The reaction is dependent from the current game mode and the game status.
Following game modes are available:

Table 4: Game modes

Game Mode	Brief Description
FOREST	Forest sequence is active
TOWER	Tower sequence is active
LOCKED	Event is happening, all controls are locked
MENU	Game Menu is active
SCORE	Highscore list is active
SHOP	Magician event is active

The Input Manager will only be active as long as the game status equals RUNNING. The game status is handled by the Games Manager.
After checking the game mode, the incoming signal will be compared and the corresponding function will be called. The input Manager is constantly called from the game loop, but only gets active if a new signal was received.

5.1.4Panel Manager

The Panel Manager takes care of all panels used in the game. All panels are generated dynamically at the beginning of the game; the Panel Manager will allocate the required memory and put all panel objects into a single linked list. To administrate the panels this way has the advantage, that all panels can be drawn with a plain and short function, regardless of the amount of panels. This structure also allows easy deletion of the panels and release of memory when the game is shut down.
The Panel Manager is called from the game loop for drawing all panels on screen. Another task of the manager is to add or set values of panels, as well as applying a short text (up to five letters) to a panel. This way the compass is realized.

5.1.5Inventory Manager

Similar to the Panel Manager, this construct administrates all inventory objects in a single linked list. This basically has the same advantages as described in 5.1.4, but on top of that, the inventory is sorted automatically. By deleting an item from a linked list, the missing spot will be filled automatically by the next inventory object. This makes any swapping and sorting algorithms, as needed by array solutions, unnecessary.
Like the Panel Manager, the Inventory Manager is called by the game loop for displaying the inventory items. There re also functions for taking and getting items available. Items of the same type are stacked automatically; the amount is not visible, because the game does not require this feature.

5.1.6Object Manager

The Object Manager is directly related to the Game Engine. After the size (zoom factor) and the horizontal position of an object were calculated by the game engine, those values are forwarded together with the map object id to the Object Manager. Then all vertices of the vector shape are calculated anew. The finished

temporary shape vertices now describe the object in a correct size and at an absolute position on screen. This information is ready to be painted on screen and the ghandler is called for drawing the object onto the display. Under some circumstances, a shape does not require to be drawn completely, because parts of it are outside the display area. Drawing those objects anyways will deliver unpredictable results, most like a system crash.
To prevent this from happening, the Object Manager provides a simple clipping algorithm. After calculating the coordinates of a shape vertex, the result is checked whether it is valid. If that is the case, the Object Manager will continue with calculating the next vertex; if the result is invalid (negative or too high values) the whole object is skipped and will not appear at all on screen.
This is a very simple algorithm that is useful for test reasons, a better but also lot more complex solution would be the Sutherland Hodgeman algorithm (see [R-2] and [R-3]).

The Object Manager is also responsible for the initialization of objects. This is necessary when new objects are created at runtime. This is the case with objects created by the Panel Manager (see 5.1.4) or the Inventory Manager (see 5.1.5).

5.1.7Message Manager

message_s
FIXED timer
UCHAR active
UINT color
char text[30]

The Message Manager is responsible for displaying text messages to the player.
A certain message can be shown in a specified color for a fixed time. Afterwards the message will be hidden automatically. The temporary data is stored in a special structure. This structure saves the time the message has been displayed, whether the Message Manager is active or not, the color of the current message, and the message itself.
The font and the time how long a message is displayed can be set with MSGFONT and MSGTIME in darktower_defines.h.
Important: MSGTIME is dependent from the initialization of the games_timer module.

The Message Manager is called by the Panel Manager, the Object Manager, the Message Manager and the Event handler.

5.1.8Level Manager

The level manager is responsible for forwarding level information to the game engine. A level consists of a two dimensional character array. For more information, take a look at 5.3.2.
Since the player only has a limited field of view, it is not necessary to scan every single array field.
According to the position of the player and the viewing angle, a certain area of the level will be taken. For each position, an entity (see 5.2.6) will be created and forwarded to the game engine. This ensures that the game is runs fast enough on the Roadrunner target, because needless calculations are eliminated that way.
The current method uses four square scan zones, which are selected according to the viewing angle of the player:

Figure 3: Scan zone 0 (45°- 135°)

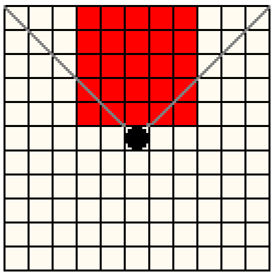


Figure 4: Scan zone 1 (135°- 225°)

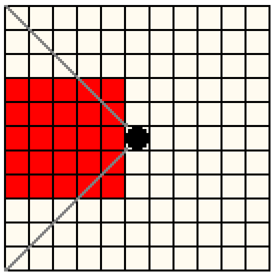


Figure 5: Scan zone 2 (225°- 315°)

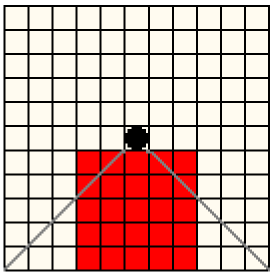
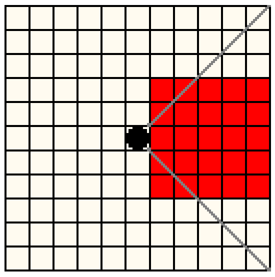


Figure 6: Scan zone 3 (315°- 45°)



As noticeable on the figures above, the array data forwarded to the game engine is critical for correct display at diagonal positions. This leaves room for later improvement.

A possible solution to this obvious problem could be defining more scan zones or introducing rectangular or even triangular scan zones.

The size of the area scanned can be determined by SCANSIZE defined in darktower_defines.h.
The level has fixed boundaries. As soon as the player comes near one of the borders, he cannot move any further into this direction and the fog event will be called. To ensure that the right memory section is accessed, the level will have a border in the size of SCANSIZE which cannot be walked into. This "dead" area is required to simulate an endless forest the player cannot escape from.

5.1.9Player Manager

player_s
UINT array_x
UINT array_y
FIXED x
FIXED y
FIXED angle

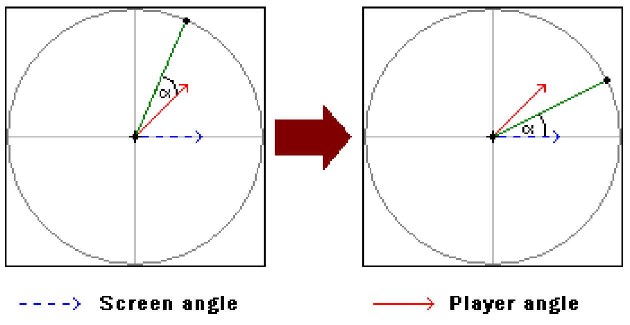
The player manager is called by the Input Manager and only gets active when any movement or rotation of the player is done. This manager is responsible for rotation, movement, acceleration and slow-down of the player. The new player position is calculated and updated. The game engine needs the player position to display the forest in 3d graphics. The player stats are stored in a special structure.
The player has to positions, the array position and the real position. This is necessary, because the player does not move from array index to array index, but makes several steps between. The array position is required by the Level Manager for generating scan zones; the absolute position is needed by the Game Engine to compute the 3d graphics accordingly.
Instead of double variables, fixed point values are used for the viewing angle and the absolute position, because of speed reasons.
In other sequences (e.g. the tower sequence) the Player Manager is not responsible for moving and rotating the virtual player, but handles selection of menu entries.

5.1.10Game Engine

The Game Engine is responsible for all 3d calculations and transformations. The object coordinates retrieved from the level array need to be converted to 2d screen coordinates. Since the game does not support different floor levels, the whole algorithm can be simplified a lot.
For all mathematical operations, fixed point mathematics is used. This has the great advantage, that long values can be used for fast calculations, while preserving a high enough precision.
The game engine receives an entity from the level manager.

The simple trick is that not the player is turning, but the objects are rotated around the player instead. This means the player always looks at 0 degrees (positive x-axis in a Cartesian coordinate system).

Figure 7: Object rotation



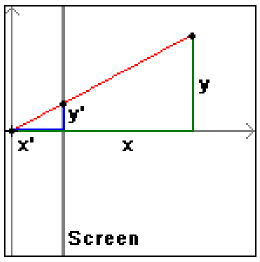
To calculate the new object positions, a rotation around the origin is required (see [R-2] and [R-3]). This can be achieved with a rotation matrix that looks like this:

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

With that method, the visual appearance will stay the very same, but the following projection algorithm will be a lot easier, because the player always looks at 0 degrees. This eliminates several sins and cosines that would be required otherwise, and the whole calculation is minimized to a quick and easy theorem on intersecting lines.

After rotating, the objects are projected onto the 2d display. The screen plane lies parallel to y'; this shows that y' must be the horizontal offset on the display.

Figure 8: Screen projection



The size of x' is a fixed value defined as VIEWDIST in darktower_defines.h and changes the horizontal distance between objects. The values x and y are calculated from the current player position and the new object position, so an equation can be arranged to resolve y':

$$y' = \frac{y}{x} \bullet x'$$

If the horizontal shift of the object is inside the display boundaries, the zoom factor needs to be calculated to display the object at correct size. With the x/y distance of this entity to the player the real distance can be determined with the Pythagoras theorem:

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

The distance equals the zoom, a factor which is required for the right scaling size by the Object Manager. This factor has to be tilted because the further away an object is, the smaller it has to be displayed, and the nearer it is, the bigger it has to be displayed. The result is multiplied with ZOOMFAC, a factor defined in darktower_defines.h. This parameter is used to control the base size of all vector objects.

The vector object contained in the entity will be retrieved and forwarded together with the horizontal shift and the zoom factor to the Object Manager.

This process is repeated until all objects within the current scan zone are transformed to screen coordinates.

5.1.11Event Handler

The Event Handler assigns events to different types of action. An event can be a lot of different things, like modifying the value of a panel, calling a special event function or changing the game mode.
The Event Handler takes care, that the right event is triggered at the right time. Since an event can happen on several occasions, the Event Handler is divided into three parts:

Table 5: Event types

Event type	**Description**
Random events	These events are executed on random occasion without having any input from the player
Object events	These events are triggered when the player collects an object
Magician events	These events are started when the player buys something in the Magician's shop

These event types will be referenced as R-, O-, and M-events from now on.

To trigger an event, the Event Handler needs an id. For O-events, this id is identical with the map object id (see 5.3.2), while for M-events this id is identical with the menu position. For R-events, an own event id was introduced.
The table below gives an overview which events exist and which event type they belong to. Some events belong to more than one event type, e.g. fog can be found in a treasure chest and can occur randomly as well.
While some events may appear belong to several event types, they are not necessarily the very same event; e.g. collecting a key costs nothing, while the magician equivalent costs several bags of gold.

Table 6: Event list

Event	Functionality	R	O	M
Fog	event function	X	X	
Plague	event function	X	X	
Magician	change game mode		X	
Brigand	change game mode		X	
Empty chest	nothing		X	
Random event	Choose random event		X	
Gain life	Panel Manager		X	X
Gain gold	Panel Manager		X	
Gain gold key	Inventory Manager		X	X
Gain silver key	Inventory Manager		X	X
Gain brass key	Inventory Manager		X	X
Gain bronze key	Inventory Manager		X	X
Gain healer	Inventory Manager			X
Gain scout	Inventory Manager			X
Gain crystal crown	Inventory Manager			X

Special events like fog and plague are directly integrated into the Event Handler.
The fog and plague events both change the game mode, and then forward their graphics to the ghandler.
After the graphic output has finished, the fog event places the player at a random position; the plague event calls the Panel Manager and decreases the number of lives by one and checks whether the amount of lives has reached '0'. If that is the case, the game mode will be changed to SCORE, the game is finished.
The random event simply chooses a random O-event id and calls the Event Manager again.

5.2Object structure

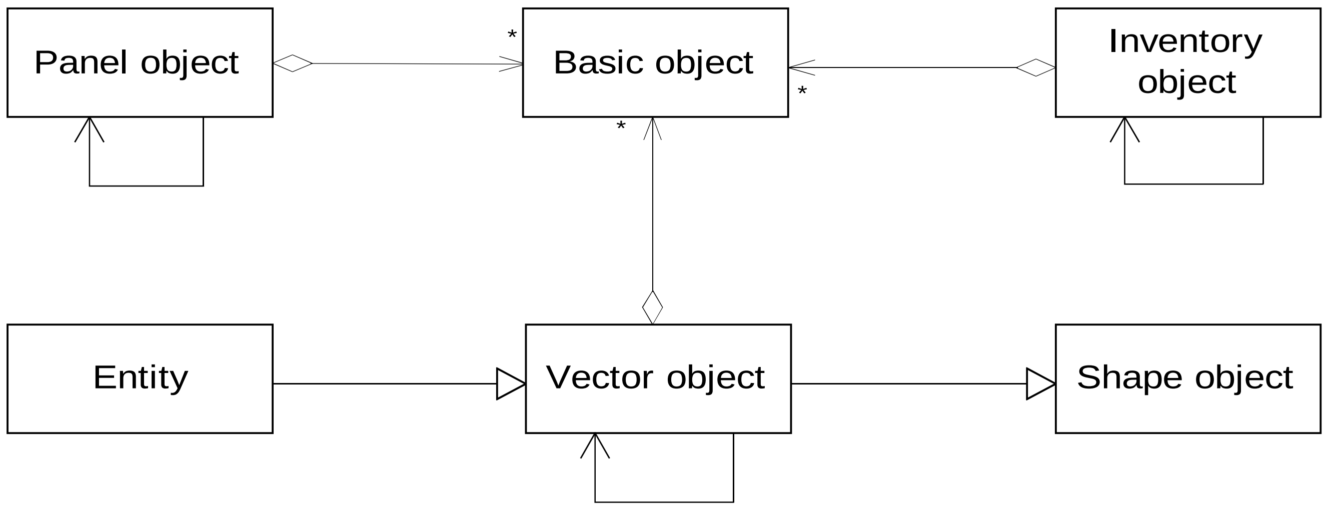
Figure 9: Object structure

Everything that can be modified, collected, moved or destroyed is internally handled as an object. Since there exist several different object types, like inventory items, OSD, 3d objects, it is required to distinguish between different object types.

5.2.1Basic object

object_s
char name[20]
short value
USHORT id
UCHAR* bmap
USHORT size_x
USHORT size_y

To get a clean structure, all objects are derived from a basic object type. This construct consists of a name (up to 20 letters), a value tag, an object id, a pointer to a bitmap (icon) and the bitmap x/y size. Not all parameters are required for all objects. A vector object which cannot be collected, most likely will not need a bitmap or a name. In that case the bitmap pointer can be set to NULL and the size can be ignored.



The value tag is only of interest for objects that can be collected. This tag determines, how many points the item is worth, also see section 6.2.5 about scoring. For all other objects, this tag is not of any interest.
Important is the object id, which is set when the object is created. This may be triggered by the inventory manager, the object manager or the panel manager. A basic object must not exist alone; it will always be part of a specific object (the specific object types are described below).

5.2.2Inventory object

inv_object_s
object_s obj
short amount
Inv_object_s* next

For managing the player's inventory, this object type was introduced. Basically, an inventory object does not differ a lot from a basic object, but it has a new parameter named amount. This parameter is used to store the number of items of one type collected by the player.
The pointer to another inventory object allows the generation of a single linked list. This makes removing the whole inventory easier, because each inventory object is allocated at runtime and memory needs to be freed as soon as the game is shut down.
Inventory objects require a bitmap for being displayed in the player's inventory.
An object name is necessary as well, because an automatically generated message will be sent to the message manager by the inventory manager when collecting or using an inventory object.

5.2.3Panel object

panel_s
object_s obj
USHORT pos_x
USHORT pos_y
char amount[7]
USHORT textpos_x
USHORT textpos_y
char font_type[10]
USHORT font_color
struct panel_s* next

The panel object is used to display information on screen, like number of lives, score, compass and the collected bags of gold. This object is more complex than the previous illustrated ones, because it also features a string display.
A panel object consists of a basic object, the x/y position on screen, a string which holds the value to be displayed, the x/y position of the string, a string to hold the font type, the font color and finally a pointer to another panel object.
The string can be used for both numeric and textual display, depending on the needs. The only thing to take care of is the limit of seven letters.
The text position is an offset to make it possible to place the text over as well as next to the bitmap. A bitmap file is optional; the panel object can be used to display some text only, as well.
The font type directly corresponds with the font file names of the graphic library; also see [R-1].
The color of the font is defined in the 8Bit color format. This color format natively comes from the graphic component and is described in 5.3.1.
Like the previously illustrated inventory object, all panel objects are arranged in a single linked list, for easier removal from memory when shutting down the game.

5.2.4Shape object

shape_s
USHORT verts
USHORT color
SH_COORD* coords
UINT center_x
UINT center_y
shtype_e type

A shape is a collection of data which can be directly forwarded to the ghandler to draw a graphic on the target display. This can be a number of lines, a single line, a filled shape, a circle or an ellipse. It represents the visual appearance of any vector object in the game.
A shape cannot exist alone; it is always part of a vector object.
This data structure contains the amount of vertices, the color of the shape, a pointer to the coordinates of each pointer, the center of the shape, as well as the shape type.
The format is illustrated more in detail in 5.3.3.

5.2.5Vector object

vec_object_s
shape_s* shp
vec_object_s* child
object_s obj
short attach_x
short attach_y

A vector object combines other vector objects and shapes together to an object, which can be displayed in vector graphics on screen. The huge advantage of vector graphics over bitmaps is the fact, that scaling and zooming as well as rotation is done a lot easier with vector objects. The downfall is, that vector graphics are less detailed, but that factor was taken into account while choosing a fitting game.
A vector object consists of a pointer to a shape, a pointer to a vector object child, a basic object, and x/y attach values.
Often, one shape is not sufficient for a vector object, because they have parts in different color or independent shapes that can't be drawn at once. To overcome this problem, vector objects can be linked together in a parent – child relation. Each vector object may have one child, and each child can have another child, and so on. This allows complex vector graphics with an easy structure. Once the last child is reached, the pointer can safely be set to NULL.
The attach values are required to align children correctly to their parent. They specify the distance between the first vertex of the parent and the first vertex of the child. The center point of the parent will be taken as center for zooming. This ensures that the shapes will always stay in the same relation, just if they were one object. For parent vector objects, the attach values have to be set to '0'.

5.2.6Entity

entity_s
UINT x
UINT y
vec_object_s* vecobj

Entities are instances of vector objects. They are temporarily generated while checking the current scan zone (see 5.1.8), and are removed as soon as they are not required by the game engine anymore. This is the case, if the scan zone changes, which happens when the player is moving or changing direction. An entity consists of x/y world coordinates (those equal the level array indices; again see 5.1.8) and a pointer to a vector object.
Entities are created by the level manager and forwarded to the game engine.

5.3Data formats

5.3.1Icon format

The icon format natively comes from the graphic component [R-1] and is used by the inventory manager and the panel manager.
A one dimensional short array holds the image information in the 8Bit color format. The first entry equals the upper left pixel of the image; the last entry equals the lower right pixel of the image.
The size of the image has to be given extra. There are *no* security routines in case the array was declared too small.
For easy creation of images, a converter tool (BMP2ICON) was written. It converts an uncompressed 24Bit Windows bitmap into the icon format.
The tool is based on Windows platforms and works in the command shell. For making the generated level info usable, it has to be converted from the Windows/DOS file format to UNIX first.

The generated include file contains data for a one dimensional character array. The size of the image is only *commented* inside and has to be defined correctly by the programmer in the appropriate C-source file.
The image data is directly used by _put_icon(), see [R-1].

The 8Bit color format is specified as shown below:

R	**R**	**R**	**G**	**G**	**G**	**B**	**B**
7	6	5	4	3	2	1	0

The highest three bits contain the red value, the following three bits contain the green value, and the last two bits hold the blue value.

5.3.2Level format

A level basically consists of a grid (array) where each field can contain one object.
Since the game engine directly calculates the object positions from the array indices, the level is stored upside down in the array, so that the positive y-axis stays the same as in a Cartesian coordinate system.
For easier creation of a good level, a converter tool (bmp2map), which reads an uncompressed 24Bit Windows bitmap file, and creates a data include file from the information, was written.
The tool is optimized to work with MS Paint. This means that the default colors defined in MS Paint have a special meaning. The converter tool translates each RGB value into a map object id (which is *not* the same as an object id). This allows generation of objects of the same object id (e.g. treasure chest, gold bag) with a different content or behavior.
The tool is based on Windows platforms and works in the command shell. For making the generated level info usable, it has to be converted from the Windows/DOS file format to UNIX first.
The generated include file contains data for a two dimensional character array. The size is commented inside.
The array declaration is *not* included and has to be written in the appropriate C-source file.
How the colors are mapped is derived from the table below:

Figure 10: Color map

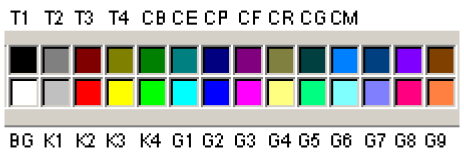

T1 T2 T3 T4 CB CE CP CF CR CG CM
BG K1 K2 K3 K4 G1 G2 G3 G4 G5 G6 G7 G8 G9

Table 7: Object Map Id

Map Object ID		Object ID		Object Name	RGB Values		
BG	0	-		Empty field	255	255	255
T1	1	TREE1	12	Maple tree	0	0	0
T2	2	TREE2	13	Elm tree	128	128	128
T3	3	TREE3	14	Pine tree	128	0	0
T4	4	TREE4	15	Dead tree	128	128	0
CB	5	CHEST	17	Treasure chest containing brigand	0	128	0
CE	6	CHEST	17	Treasure chest containing nothing	0	128	128
CP	7	CHEST	17	Treasure chest containing plague	0	0	128
CF	8	CHEST	17	Treasure chest containing fog	128	0	128
CM	9	CHEST	17	Treasure chest containing magician	128	128	64
CG	10	CHEST	17	Treasure chest containing gold/warrior	0	64	64
CR	11	CHEST	17	Treasure chest containing random event	0	128	255
K1	12	GOLDKEY	1	Golden key	192	192	192
K2	13	SILVERKEY	2	Silver key	255	0	0
K3	14	BRASSKEY	3	Brass key	255	255	0
K4	15	BRONZEKEY	4	Bronze key	0	255	0

G1	16	GOLD	8	Bag of gold (1)	0	255	255
G2	17	GOLD	8	Bag of gold (2)	0	0	255
G3	18	GOLD	8	Bag of gold (3)	255	0	255
G4	19	GOLD	8	Bag of gold (4)	255	255	128
G5	20	GOLD	8	Bag of gold (5)	0	255	128
G6	21	GOLD	8	Bag of gold (6)	128	255	255
G7	22	GOLD	8	Bag of gold (7)	128	128	255
G8	23	GOLD	8	Bag of gold (8)	255	0	128
G9	24	GOLD	8	Bag of gold (9)	255	128	64
Invalid color			255	Error in map, ignored by level manager	-	-	-

5.3.3Shape format

shape_s
USHORT verts
USHORT color
SH_COORD* coords
UINT center_x
UINT center_y
shtype_e type

A shape is a collection of data which can be directly forwarded to the ghandler to draw a graphic on the target display. This can be a number of lines, a single line, a filled shape, a circle or an ellipse.

- *verts* specifies the number of vertices the shape has.
- *color* is needed for determining the shape color in 8Bit format.
- *coords* is a pointer to an GH_COORD array, containing x,y values for each vertex.
- *center_x, center_y* specify the center of the shape. This is important for the vertical and horizontal offset when scaling the shape
- *type* includes the type of shape, possible are
  - POLY: filled polygonal shape
  - LINE: a single line
  - POLYLINE: a series of lines linked together
  - CIRCLE: a filled circle
  - ELLIPSE: a filled ellipse

GH_COORD
USHORT x
USHORT y

Some extra rules apply to circles and ellipses, as described in the following part.
Number of vertices *must* be set to '1'.
The first vertex is interpreted as center of the circle or ellipse.
For a circle, the x-value of the second vertex is interpreted as radius, the y value is ignored.
For an ellipse, the second vertex represents the radius in x and y direction.

The 8Bit color format natively comes from the graphic component and is described in 5.3.1.

6 Game manual

6.1Game objective

The player is exposed to a forest full of dangers and on the hunt after four rare keys, that are hidden somewhere within this forest. These are required to solve the riddle of the Dark Tower.
While searching for the keys, the player will stumble over a lot of different treasure chests, which may contain valuable treasures – as well as the deadly plague or the confusing fog.
If the puzzle of the tower can be solved before all lives are lost, the game is finished. Independent from whether the game was finished successfully or not, the final advance will be measured by a total score.

6.2Rules of the game

6.2.1Controls

There are two different game modes, the Forest mode, and the Tower mode. The Forest sequence will be active most of the time, the Tower sequence will only become active once the player collected all four keys and found the tower.

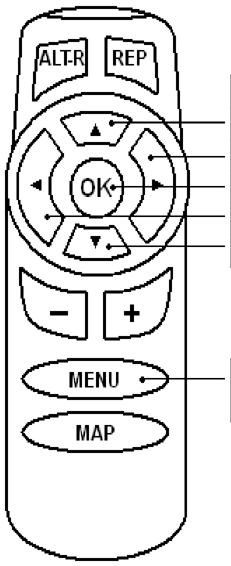


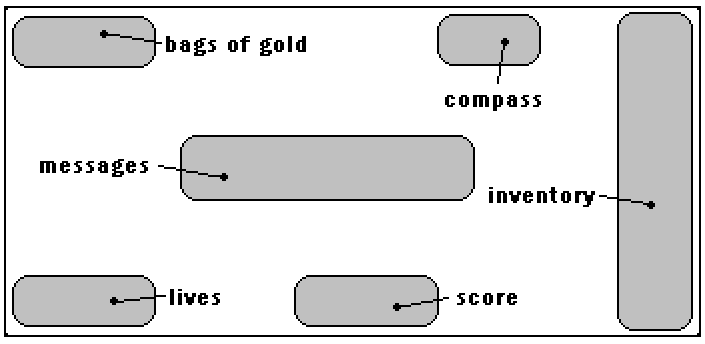
Table 8: Game Controls

Forest Sequence	Tower Sequence
Move forward	Select keyhole
Turn right	
Grab/Use objects	Select key (press multiple times)
Turn left	Select keyhole
Move backward	

Return to Games Manager	Finish sequence

There is a lot of information displayed on the screen. Messages will stay for a short time only, all other panels and the inventory will stay visible for the whole game.
This is what the player's screen looks like:

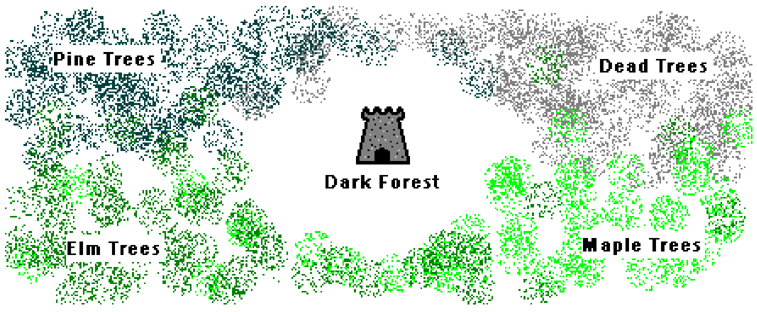
Figure 11: Player screen



6.2.2Forest sequence

The forest is divided into four equal sectors, each represented by a different type of tree - pines, elms, maples, and dead trees. In the very center of the forest, called the Dark Forest, there are no trees at all. This is where the Dark Tower is hidden and will become visible only when the player has collected all four keys (see 6.2.4). There are also smaller dark forest zones scattered throughout the forest map. It is easy to get lost in these zones.
In the Zone of Death, the plague occurs more often. It is advisable to enter the Zone of Death only when the player has a healer in his possession (see 6.3). If the player reaches the boundaries of the forest, the trees disappear, a fog appears and the player is placed randomly somewhere else within the forest.

Figure 12: The Dark Forest



The direction buttons of the remote to move the player around the forest. As the player walks through the forest, a compass shows his direction of travel at the top right side of the screen.

In addition to the boundary fog, the player can also run into fog while walking within the forest. The player may also find fog in some of the chests scattered throughout the forest (see 6.3). When the player runs into fog, he will be placed randomly somewhere else within the forest unless he has a scout in your possession (see 6.3).

Like fog, the player can run into the plague while walking throughout the forest and may also find the plague in a chest (see 6.3). When the player runs into the plague, he will lose a life unless he has a healer in his possession (see 6.3).

6.2.3Magician sequence

The Magician Sequence begins as soon as the player opens a magician chest. If the player has enough gold in his inventory, the magician may sell a key, an extra life, a healer, a scout or the crystal crown.
The amount of gold needed to buy these treasures is as follows:

Table 9: Magician prices

Treasure	Gold Needed
Gold Key	60-69 bags
Silver Key	50-59 bags
Bronze Key	40-49 bags
Brass Key	30-39 bags
Extra Life	10-20 bags
Crystal Crown	60-69 bags
Healer	15-20 bags
Scout	20-30 bags

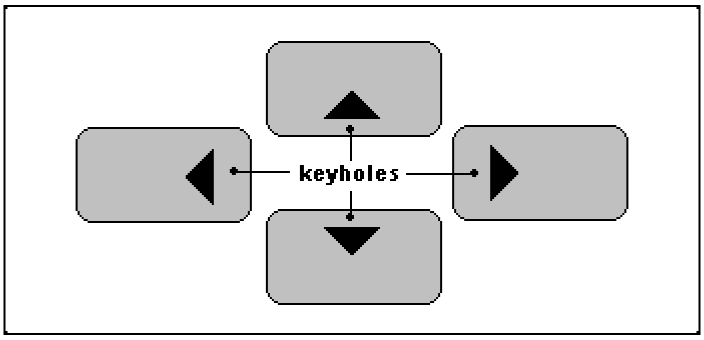
The crystal crown is a special treasure that is worth 1500 points if it is in the player's possession at the end of the game. The healer is a helper who prevents the fatal effects of the plague. The healer can be used only 4-6 times while in the player's possession and will then be lost.

Another healer may be granted by the Magician later in the game if there are still magician chests available and the player has enough gold left.
A scout is another helper. The scout prevents the player from getting lost in a fog. The only exception is the boundary fog. As with the healer, the scout can be used only 4-6 times while in the player's possession. Another Scout may be granted by the magician later in the game if there are still magician chests available and the player has enough gold left.

6.2.4Tower Sequence

Once the player has all four keys in his possession, he can enter the tower and try to solve the riddle of the keys to open the tower door. The tower is visible only when all four keys are in the player's possession. It is located near the center of the map, which is surrounded by a mixture of all four types of trees. When the player finds the tower, he can enter the tower by pressing the OK button.
Once the player is inside the door, he must put the four collected keys into the keyholes in the correct order to try and unlock the tower. The desired keyhole is selected with the direction buttons. To select the type of key (gold, silver, bronze or brass) press OK one or more times. Each key should only be used once.
The Tower sequence is shown in the following figure:

Figure 13: Tower sequence



The solution is tested by pressing the MENU button. If all four keys are in the correct position, the game will end. If not, those keys that are in the correct positions will stay lit for a few seconds and then the player will be placed back into the forest at a randomly selected location. If the player has a Scout in his possession, he will be placed outside the tower rather than somewhere within the forest, and he can enter the tower and try to solve the riddle of the keys again.

6.2.5Scoring

Points are awarded for each treasure the player picks up after the start of the game. Score also increases for solving the riddle of the Tower.

Possessions are awarded as followed:

Table 10: Scoring

Item	Award
Gold Key	1,000 points
Silver Key	900 points
Bronze Key	800 points
Brass Key	700 points
Bag of Gold	100 points
Lives Left	100 points
Crystal Crown	1,500 points
Healer	300 points

Scout	300 points
Destroying each Brigand	125 points
Solving the Riddle of the Tower	3,000 points

6.3List of items

There are a number of items that appear throughout the game in the forest sequence. Items can be collected or opened by walking towards them and then pressing the OK button.
The following table shows all available items and their use:

Table 11: List of items

Item	Description
Gold Key Silver Key Brass Key Bronze Key	There is a key hidden in each of the four sectors of the forest. Once the player collected all keys, he can proceed with the search for the dark tower, which is only visible when the player is in possession of all four keys. The keys are also available at the magician's shop.
Bag of Gold	Scattered throughout the forest are bags of gold which the player can pick up and add to his possessions. They Bags of gold can only be collected once.
Healer	The healer is a helper who prevents the player from getting lost in a fog. The only exception is the boundary fog. The healer can be used only 4-6 times while in the player's possession and will then be lost. Available at the magician's shop.
Scout	The scout is a helper who prevents the fatal effects of the plague. The healer can be used only 4-6 times while in the player's possession and will then be lost. Available at the magician's shop.
Crystal Crown	A valuable treasure which will boost up the player's score when finishing the game. Available at the magician's shop.
Treasure Chest	There exist six different types of treasure chests: • *Treasure Chests:* These chests are very rare and contain a no-risk treasure (one to nine bags of gold or one to four extra lives). A particular treasure chest can only be opened once per game. Opening the same chest again will have no further effect. • *Empty Chests:* These are empty chests that will open, pause for a moment, and then close again. • *Fog Chests:* These chests contain fog. When fully opened, the fog event will occur, as explained earlier. • *Plague Chests:* These chests contain the plague. When fully opened, the plague event will occur, as explained earlier. • *Brigand Chests:* These contain enemy beasts called 'Brigands' who the player must fight for survival (feature not set in stone yet and subject to change). • *Magician Chests:* These contain magicians who can provide the player with many valuable treasures (see 6.2.3). As with treasure chests, each magician chest can only be opened once per game. Opening the same chest again will have no further effect.

7 Remarks

- Document does not yet cover the implementation of a highscore list based on the OPA concept
- The new games manager, which should consist of a menu to select a game, is not yet covered in this specification
- While the document covers game rules for the brigand sequence, it is not yet sure if time factors will allow implementing this sub game into the final game.