GamesCrafters 2006

Using Tier Gamesman

Tutorial Documentation

Max Delgadillo (Dev)

Deepa Mahajan (Dev)

Yanpei Chen (Doc)

Revision Summary			
Author	Date	Version	Comments
Yanpei Chen	2006.09.27	0.1	Created
Yanpei Chen	2006.10.17	0.2	Revisions after initial doc review
Yanpei Chen	2006.12.20	0.3	Additions to list of games with tiers implemented

Table of Contents:

1. Overview

Tier Gamesman is a significant extension to the original Gamesman. It breaks up the set of all game positions into subsets called tiers. Each tier is solved independently, and the results for each tier is merged together to form the solution for the entire game. Tier Gamesman offers several breakthrough functionalities:

- Solving large games tier by tier instead of all at once.
- Less memory, shorter latency to solve each tier.
- Different tiers solved concurrently in parallel on many machines.
- Bottom up solving, traversing the game tree from primitive positions.

In addition, there are several more positives:

- Completely compatible with original Gamesman core API.
- Simple API for game modules to add tier solving capabilities.
- Feature set completely independent of other functions in game modules.

Games already solved using Tier Gamesman:

- Tic-Tac-Toe
- Bagh Chal
- Quick Chess
- Abalone
- Six Men’s Morris
- Mancala

Games with ongoing tier work:

- Quarto!
- Tile Chess
- Cambio
- Ataxx

Games that can be solved with future tier work:

- Connet-4
- Othello

The above lists are by no means exhaustive, and the lists are expanding with increasing energy devoted to tier work.

2. Goals and Non-Goals of this Document

This documentation target developers who are seeking to add tier functionalities to existing game modules. Developers should walk away with the following:

- High level system overview of Tier Gamesman.
- Familiarity with the API between Tier Gamesman and the game module.

- Reference to an example implementation of tier functionalities.
- Confidence to begin their own implementation of tier functionalities.

Readers of this documentation are expected to have familiarity with C, the high-level Gamesman system architecture, and the API between Gamesman core and modules. In particular this documentation will not

- Offer tutorial on C.
- Offer tutorial on Gamesman architecture or other extensions to Gamesman.
- Offer tutorial on how to implement other parts of the game modules.
- Detail how Tier Gamesman solves and merges results.

It is not necessary to have an in-depth, below-the-API understanding of Tier Gamesman to implement Tier functionalities in game modules.

All other issues not listed as goals would be considered non-goals.

3. Breaking up a Game into Tiers

There is only one constraint for breaking up a game into tiers: The resulting tier tree should be a directed, acyclic graph. In other words,

Tiers must not be loopy.

As one traverses the game tree according to the rules of the game, one cannot re-enter a given tier after exiting from it earlier. Namely, loops such as Tier1 – Tier2 – Tier1, or Tier1 – Tier2 – …… – TierN – Tier1 must never occur.

However,

Each tier may be loopy.

In other words, as one traverse the game tree according to the rules of the game, one may revisit a position examined earlier, so long as traversal remains within the same tier at all steps between the initial traversal and the revisit.

Tiers may be defined in many ways.

Many games lend themselves to easy break up into tiers. Tiers may be defined by

- Pieces on board.
- Moves played.
- Game scores.
- Pieces captured.

The list above is suggestive and not exhaustive.

Also,

Tier definitions must remain constant.

Where many tier definitions are possible, tier definition cannot change during the course of solving/playing the game. Here, the game module must choose one of the several possible tier definitions, and stick with it throughout the course of solving/playing the game.

4. API Between Tier Gamesman and Game Modules

4.1 API Overview

The API to facilitate Tier Gamesman involves several new data types:

- `typedef unsigned int TIER;`
- `typedef POSITION TIERPOSITION;`
- `TIERLIST* CreateTierlistNode(TIER, TIERLIST*);`

There are also several essential functions that require the game module to adhere to naming and function prototype:

- `TIERLIST* (*gTierChildrenFunPtr)(TIER);`
- `TIERPOSITION (*gNumberOfTierPositionsFunPtr)(TIER);`
- `void gUnhashToTierPosition(POSITION, TIERPOSITION*, TIER*);`
  `POSITION gHashToWindowPosition(TIERPOSITION, TIER);`

Modifications to the existing hash()/unhash() functions are also necessary.

The game module may provide module-specific support functions, where the module may name the support functions whatever it likes. Support functions may include:

- `TIER GetTier (Board*);`
- `TIERPOSITION TierHash (Board*);`
- `Board* TierUnhash(TIERPOSITION);`
- `POSITION HashWithoutTiers (Board*);`
- `Board* UnhashWithoutTiers (POSITION);`

In addition, there are global variables that need to be set:

- `extern BOOLEAN kSupportsTierGamesman;`
- `extern BOOLEAN kExclusivelyTierGamesman;`
- `extern BOOLEAN kDebugTierMenu;`
- `extern TIERPOSITION gInitialTierPosition;`
- `extern TIER gInitialTier;`

Lastly, the module may also implement optional functionalities, where the naming and function prototype must be adhered to again:

- `extern BOOLEAN (*gIsLegalFunPtr)(POSITION);`
- `extern STRING (*gTierToStringFunPtr)(TIER);`

4.2 New Data Types

There are several data types introduced by Tier Gamesman.

```
typedef unsigned int TIER;
typedef POSITION TIERPOSITION;
TIERLIST* CreateTierlistNode(TIER, TIERLIST*);
```

The definitions of the TIER and TIERPOSITION types appear in gamesman/src/core/types.h.

4.3 Global Variables

Game modules need to be aware of several global variables:

- **`extern BOOLEAN kSupportsTierGamesman;`**
  Set to TRUE when the game module implements Tier Gamesman.
- **`extern BOOLEAN kExclusivelyTierGamesman;`**
  If set to TRUE, Tier Gamesmann will be used no matter what. The behavior of setting this variable when tiers are not implemented is undefined.
- **`extern BOOLEAN kDebugTierMenu;`**
  If set to TRUE, starting to solve the game will start the dedicated Tier Gamesman debugger instead of the solver. The Tier Gamesman debugger will help debugging all the tier API functions.
- **`extern TIERPOSITION gInitialTierPosition;`**
  The TIERPOSITION upon entry into Tier Gamesman. Along with gInitialTier, it is equivalent to gInitialPosition in the original Gamesman.
- **`extern TIER gInitialTier;`**
  The TIER upon entry into Tier Gamesman. Along with gInitialTierPosition, it is equivalent to gInitialPosition in the original Gamesman.

The game module sets these values upon initializing the game.

4.4 Essential Functions

To leverage tier functionalities, game modules need to implement several functions:

```
TIERLIST* (*gTierChildrenFunPtr)(TIER);
```

The game module provides a function that takes in a tier argument, and returns a list of tiers that are immediately reachable from the tier argument. In other words, the returned list contains all tier "children" of the argument tier.

The game module is responsible for referencing the gTierChildrenFunPtr to an actual function with the correct prototype. The game module may arbitrarily name the referenced function. The game module is responsible for implementing the referenced function.

gTierChildrenFunPtr is defined in gamesman/src/core/globals.c, and is initialized to NULL.

```
TIERPOSITION (*gNumberOfTierPositionsFunPtr)(TIER);
```

The game module provides a function that takes in a tier argument, and returns the highest TierPosition value (ideally, equal to the number of tierpositions) associated with that tier.

The game module is responsible for referencing the gNumberOfTierPositionsFunPtr to an actual function with the correct prototype. The game module may arbitrarily name the referenced function. The game module is responsible for implementing the referenced function.

gNumberOfTierPositionsFunPtr is defined in gamesman/src/core/globals.c, and is initialized to NULL.

```
void gUnhashToTierPosition(POSITION, TIERPOSITION*, TIER*);
POSITION gHashToWindowPosition(TIERPOSITION, TIER);
```

This pair of functions provides the TIERPOSITION + TIER ⇔ POSITION translation. The game module must adhere to the naming and the prototype provided.

These functions guarantee only to return some POSITION when given a TIERPOSITION and a TIER, and return some TIERPOSITION and some TIER when given POSITION. The game module is responsible for book-keeping among tiers, switching hash contexts, and making sure the semantics of tier and tier position are correct.

If the input to gHashToWindowPosition is not a valid TIERPOSITION in the context of the particular tier (generates an illegal POSITION), gIsLegalFunPtr must be set to a user-defined function checking for valid POSITIONs.

The functions require below-the-API initialization. Compete initialization may be checked by accessing the global gHashWindowInitialized Boolean.

4.5 Essential Modifications – Hash/Unhash

Hash and unhash are not a part of the mandatory API for the original Gamesman; however they are needed for all but the trivial games. Hash and unhash implementations depend completely on each game module.

Hash() is needed both when solving with the original Gamesman and with Tier Gamesman. Thus hash() needs to be backward compatible.

Hierarchical Hash for Tiers

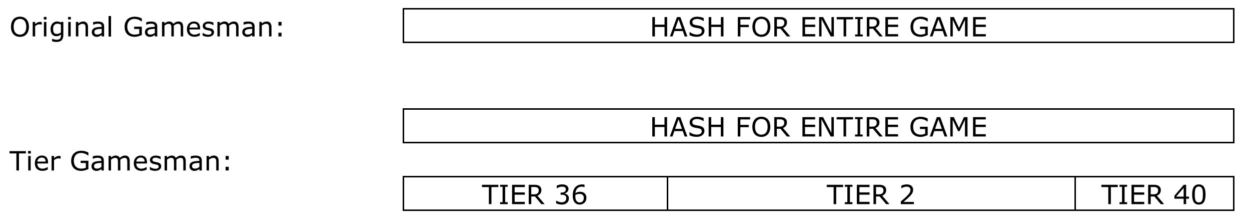
With Tier Gamesman, hash breaks up into two levels:

1. Hash for the entire game, with hash values spanning all possible positions.
2. Hash for each tier, with hash values spanning all possible positions within a tier.

These levels transform the flat hash space associated with the original Gamesman into a hierarchical hash space.

- Hash into POSITION for the entire game.
- Hash into TIERPOSITION for each tier.

The TIERPOSITION hash must be as compact as possible to ensure efficient solving by Tier Gamesman.



Hierarchical hash needs translations from TIERPOSITION to POSITION.

To ensure backwards compatibility, there need to be two hash algorithms:

- For original Gamesman: Board → POSITION.
- For Tier Gamesman: Board → TIERPOSITION → POSITION.

In particular, game modules should initialize BOTH the tier hashes and the original hash for the entire game. The original hash for the entire game should be the active hash context when InitializeGame() or GameSpecificMenu() returns.

Here we provide pseudo code guides to both facilitate Tier Gamesman and ensure backward compatibility with the original Gamesman.

POSITION hash(Board);

Possible implementation of hash() may be

```
POSITION hash(Board) {
      POSITION position;
      if (using Tier Gamesman) {
            TIER tier = GetTier(board);
            Switch tier hash context;
            TIERPOSITION tierPos = TierHash(board);
            position = gHashToWindowPosition(tierpos, tier);
      } else position = HashWithoutTiers(board);
      return position;
}
```

The condition "if (using Tier Gamesman)" is implemented as "if (gHashWindowInitialized)". gHashWindowInitialized is a global variable set and unset below the Game Module – Tier Gamesman API. Game modules only need to know that gHashWindowInitialized is the Boolean equivalent to "using Tier Gamesman", and game modules must not set and unset gHashWindowInitialized.

Functions named in the above pseudo code will be explained later.

Board unhash(POSITION);

Possible implementation of unhash() may be

```
Board unhash(POSITION position) {
      if (using Tier Gamesman) {
            TIERPOSITON tierpos; TIER tier;
            gUnhashToTierPosition(position, &tierpos, &tier);
            Switch tier hash context;
            return (Board *) TierUnhash(tierpos);
      } else return UnhashWithoutTiers(position);
}
```

The condition "if (using Tier Gamesman)" is implemented as "if (gHashWindowInitialized)". gHashWindowInitialized is a global set and unset below the Game Module – Tier Gamesman API. Game modules only need to know that gHashWindowInitialized is the Boolean equivalent to "using Tier Gamesman", and game modules must not set and unset gHashWindowInitialized.

Functions named in the above pseudo code will be explained below.

Necessary Support Functions:

- **`TIER GetTier (Board*);`**

  Board* is a game-module specific data structure representing the game board. This function takes in any possible game board, and return the tier associated with the board. Each game module may define tiers differently. The naming and implementation of this function is completely up to each game module.

- **`TIERPOSITION TierHash (Board*);`**
  `Board* TierUnhash(TIERPOSITION);`

  This pair of functions provides the Board ⇔ TIERPOSITION translation. The naming and implementation of this function pair is completely up to each game module.

  Multiple hash contexts may exist, one context associated with each tier. The game module is responsible for switching between hash contexts for different tiers.

  Games using the generic hash libraries may leverage the context switching functionalities to be found there.

- **`POSITION HashWithoutTiers (Board*);`**
  `Board* UnhashWithoutTiers (POSITION);`

  Copy paste of the hash/unhash functions written for the original Gamesman. The naming and implementation of these functions are completely up to each game module.

4.6 Optional Functions

In addition to the essential functions and modifications above, game modules may choose to implement the following optional functions.

- **`extern BOOLEAN (*gIsLegalFunPtr)(POSITION);`**
  NOTE: This function is required if there are illegal tier positions. Optimizes Tier Gamesman solving by pruning the tiers for illegal positions.

- **`extern STRING (*gTierToStringFunPtr)(TIER);`**
  Returns a human readable text string describing the tier, e.g. "Tier 9: 9 pieces on board".

5 Example Implementation

An example implementation is found in gamesman/src/mtttier.c.

This module implements Tic-Tac-Toe for the Tier Gamesman. It is also completely compatible with the original Gamesman.

mtttier.c adheres rigorously to the prototypes for

- `TIERLIST* (*gTierChildrenFunPtr)(TIER);`
- `TIERPOSITION (*gNumberOfTierPositionsFunPtr)(TIER);`
- `void gUnhashToTierPosition(POSITION, TIERPOSITION*, TIER*);`
- `POSITION gHashToWindowPosition(TIERPOSITION, TIER);`

It also sets the global variables

- `BOOLEAN gUsingTierGamesman;`
- `TIERLIST* gTierSolveListPtr;`
- `TIERPOSITION gInitialTierPosition;`
- `TIER gInitialTier;`

All other tier functions/variables introduced in mtttier.c are module-specific naming and implementations.

6 Running Tier Gamesman

After game modules implement tier functionalities, set gUsingTierGamesman to TRUE, and game module compiles, the tier functionalities can be accessed from the Gamesman text UI:

1. Start Gamesman.
2. In the main menu, set to use Tier Gamesman using the (t) option.
3. Use the (w) option to play the game using current tier settings.
4. Use (s) to solve. This will enter the Retrograde Solver menu, with straightforward interface.

All other interfaces in the Gamesman text UI are unchanged.